metricSplit–
Automated Notation of Rhythms, Adequate to a Metric Structure
TECHNICAL REPORT

Markus Lepper
semantics GmbH

Baltasar Trancón y Widemann
Programmiersprachen und Compilertechnik
Technische Universität Ilmenau

2017-03-31

urn:nbn:de:gbv:ilm1-2017200259

TECHNISCHE UNIVERSITÄT
ILMENAU

Zusammenfassung

In der traditionellen „Westeuropäischen Standard-Notation“ der Musik („Common Western Notation“, CWN) kann jeder als Folge von Dauernwerten gegebener musikalischer Rhythmus mittels der grundlegenden Mittel der Dauernbeschreibung (Notensymbole, Verlängerungspunkte, Proportionalklammern, Haltebögen) auf unendlich viele Weisen dargestellt werden.

Ein musikalisches Metrum ist eine hierarchische Gliederung der Dauer eines Taktes in Unterintervalle. Ein Metrum schränkt die Anzahl der möglichen Notationen deutlich ein auf die ihm adäquaten. Dies bedeutet: ergonomisch zweckmäßig für leichte Erfassbarkeit der ryhthmischen Gestalt als solcher und ihrer Stellung relativ in der Hierarchie des Metrums.

Im Falle automatisch generierter Dauernfolgen, z.B. durch algorithmische Komposition oder computergestützte Analyse, erhebt sich das Problem der automatischen Berechnung solch adäquater Darstellungen. Dies erledigt der Algorithmus METRICSPLIT, der in dieser Schrift spezifiziert wird. Als *vollständige* Funktion über alle möglichen Eingaben ist er ohne veröffentlichte Vorläufer. Die zu seiner Konstruktion und Beschreibung notwendigen Formalisierungen sollen auch beitragen zur allgemeinen Diskussion über Rollen und Eigenschaften von mathematischen Modellen in ästhetischen Kontexten.

Abstract

In traditional Common Western (music) Notation (CWN), for any *musical rhythm* given as a sequence of duration values, there are infinitely many possibilities for its notation, using the standard devices: note symbols, prolongation dots, ties and proportional brackets. But given additionally a *musical metrum*, which in CWN is some hierarchical organization of time points, repeated with every measure, this number is significantly reducible to those few which are *adequate* to both, metrum and rhythm, which means to be ergonomically sensible for easy reading and for immediately grasping the rhythm in relation to the structure of the metrum. In case of automatically generated rhythms, e.g. by algorithmic composition or analysis, the problem arises of finding the adequate notation automatically. This is achieved by the algorithm METRICSPLIT, presented in this paper. As a total function, it is without a published predecessor. The necessary analysis and modeling shall also serve as a contribution to theoretical discussion.

Contents

List of Figures

List of Tables

Acknowledgment

The music engraving in the examples has nearly completely been realized by LilyPond v.16.0, [6]. (At very few places, beams have been painted over manually.)

1 Definition of the Problem

1.1 Scope of the metricSplit Project

METRICSPLIT is an algorithm to transform an arbitrary sequence of music duration values into a rhythmic notation, adequate to a given metric structure, in the sense of traditional *Common Western Notation (CWN)*. It thus realizes a *total function* from abstract duration values and parameter settings into a sequence of CWN notation symbols. To the best of our knowledge, this is the first published such algorithm.

METRICSPLIT has been invented and is best suited for rendering the results of computer aided composition. But it may also serve for historic research by reconstruction: Searching for the parameter settings which reproduce a particular historic notation practice may shed light on their characteristics.

The algorithm is subject of ongoing research; esp. the set of style parameters will likely be expanded. The state described here is of **20170331**.

The algorithm is currently implemented in Java. An simple *interactive demonstration tool* can be downloaded from

http://bandm.eu/metatools/download/DemoMetric.jnlp

It takes the rhythmic contents of one single measure and a metric tree specification (as explained below) and delivers the sequence of music notation duration symbols in plain text and as LILYPOND source text [6]. The Java class code contained may be employed in user applications, under the CC-NCBYSA license; for this purpose see the API doc at http://bandm.eu/music/docs/api.

1.2 Intention and Structure of this Article

The rest of this first section discusses the problem of rhythmic notation in CWN, the roles of algorithms in arts and humanities in general, as far as relevant, and the resulting requirements. Also different meanings of "adequate" in the title of this article will be substantiated.

Section 2 gives a specification of the METRICSPLIT algorithm, mostly in natural language, w.r.t. the intended audience. Formalization has only been applied due to convincing cost-benefit ratio, e.g. for grammars.

1.3 CWN, Rhythm, Measure and Metrum

Common Western Notation (CWN) is used in the following text for the traditional rules for notating music, developed from the seventeenth century up to now, mainly in the European context. CWN is among others characterized by the fact that rhythms are defined (i.e. composed, notated and performed) relative to a sequence of *measures*. A measure is defined as a sub-interval of a piece's playing time. Each measure adheres to a certain *metrum*. The piece's playing time is a sequence of adjacent measures, and the metrum in most cases stays the same for a large number of measures, mostly even for the whole piece. A metrum can be seen as a selection of distinct time instances relative to the start of the measure, together with a mapping of a *metric role* to them. A *(musical) event* can only occur at one of these time points. All time points of a metrum have a specific relation to their neighbours, and they give different importance to the occurring events.

The exact meaning of these notions "role", "relation", "importance", is widely varying with genre, epoch and style. E.g., for a particular family of dances, events at different time points of the measure are to be played with different intensity; or certain changes of harmony may only happen at selected time points, etc. Furthermore, each metrum normally reflects the psycho internal mechanisms of "counting and acting" by the executing musician, and the sequence of measures and of time points relative to a measure, together are the means for orientation and synchronization. For the following considerations all these concrete definitions of metric roles are out of scope; only the mere fact that they do exist is fundamental.

The selection of distinct time points is equivalent to subdividing an interval of time. This principle can be applied recursively: Downwards each level of sub-intervals can be further subdivided, and upwards more than one measure can be aggregated. This results in a *metric hierarchy*, which is the direct cause for the specific mechanisms for rhythmic notation in CWN.

The historic evolution of notation happened as a chain of small steps, most of them "self-explaining", because they shifted a well-proven notational device only slightly sideways, into a new application context. E.g., the *dotted notation*, invented originally as a kind of short-hand notation for one particular rhythmic pattern "three plus one", emancipated to a stand-alone notation for the "factor three". These steps result in an overall tendency for *increasing abstraction*, *generalization* and *compositionality* of music notation. Nowadays, this has the effect that dances from the seventeenth century as well as Darmstadt-style avantgarde music can be notated by basically the same symbols and interpretation rules.

We see two fundamentally different roles for metrum in contemporary CWN notation: Its origin is in dance music, with different metric roles and characteristics for different time points, see above. This variant is called *expressive metrum* in the following: the metrum per se carries some "meaning".

The other extreme uses metrum only for notating durations and organizing execution, *without* any metric roles beyond mere synchronization purpose. In Darmstadt-style, e.g., a composer may construct a piece in seconds or milliseconds or rational numbers, and finally just notate these values in traditional notes, using a 4/4 metrum with MM=60, for mere convenience. This we call a *non-expressive metrum*.

Normally the mapping from a metrum to physical time includes some kind of distortion in the expressive case, but not in the non-expressive.

There are very different concrete calculation tasks in the field between rhythm and metric:

- **(TNoteSymbs)** For a given metrum and a given sequence of duration values (given in some abstract encoding, e.g. as mere rational numbers), a sequence of *duration symbols* from musical notation must be found, which is adequate to both.
- **(TSpont)** For a sub-sequence of durations which do not fit into the grid of selected time points pre-defined by the metrum, an appropriate extension is added to the definition of the metrum automatically.
- **(TInfer)** From a sequence of durations and no metrum indication at all (or perhaps only a given measure duration) a metrum must be *inferred* which represents the metric structure of the event sequence in an optimal way.
- **(TInferPlus)** Here the inference of metra is not carried out on sequences of pure duration values only, but with *additional input* like intensity and stress patterns, instrumental polyphony and coincidence, harmonic changes, etc. In

different contexts of musicology, this variant of the preceding task is of great interest, mostly not related to the question of notation, but to perception, style classification, ethnology, physiology, etc.

The main task in case of an expressive metrum is **(TNoteSymbs)**. Since METRICSPLIT supports user-defined variants in the metric specification, from which it selects automatically, the user can foresee all metric alternatives (like triplets on different levels) and precisely control their appearance.

When rendering complex nested divisions out of an algorithmic composition process into a non-expressive metrum, there may be too many combinations of nested n-plets to be foreseeable. An algorithm could react automatically and "fix the hole" without the user's intervention. This is covered by task **(TSpont)**.

These both tasks arise e.g. when results of algorithmic composition or of automated transformation or analysis shall be rendered into CWN notation automatically and are both covered by METRICSPLIT.

Far further go tasks **(TInfer)** and **(TInferPlus)**, because they heavily rely on empirical data from physiology, psychology, acoustics, culture, history, etc., widely varying with the application context. These tasks form a research area of their own, not covered by our work, and are included in the list above for separation only.

The tasks **(TNoteSymbs+TSpont)** are more complicated than they might seem, because very *different requirements* come together:

- **(RqArith)** The *arithmetic* requirement simply says that the durations of the measure as a whole and of its sub-intervals, as defined by the metrum, must be numerically equal to the durations of the rhythm notated therein, based on the durations of the note symbols.
- **(RqErgo)** The *ergonomic* requirement says that the chosen variant of notation should show as clear as possible (a) the metric structure of the measure as such, and also (b) the roles of all events occurring, i.e. their positions relative to the metrum's hierarchy.
- **(RqStyle)** The *stylistic* requirement realizes the fact that with epoch, genre and style different variants of notation have been preferred and are more common than others, due to historic development.
- **(RqInt)** The *intentional* requirement includes that the author of the notated work may want to express further information, beyond the mere duration, e.g. about rhythmic character, articulation, motive identity, or even on a conceptual level.

This list follows decreasing degree of possible automation: It is trivial for the first and by definition impossible for the last.

Please note that **(RqErgo)** has different consequences when applied to an expressive and to a non-expressive metrum. In the first case the selection of the metrum is fixed by the composer to indicate the character of the piece, and even most complex rhythms may not change the metrum fundamentally. E.g. in the piano suites by BACH the complex rhythmic foreground structures can show a big distance to the original simple dance metrum they pretend to realize.

But in the non-expressive case, the metrum only serves the technical purposes of execution and synchronization, and arbitrary switches between very different metric structures can be an adequate means to achieve the ergonomic requirement.

There are cases in which requirements conflict. The first example in Figure 1 starts at bar number 19 of Cp XVI of "DIE KUNST DER FUGE" (GRÄSER edition).

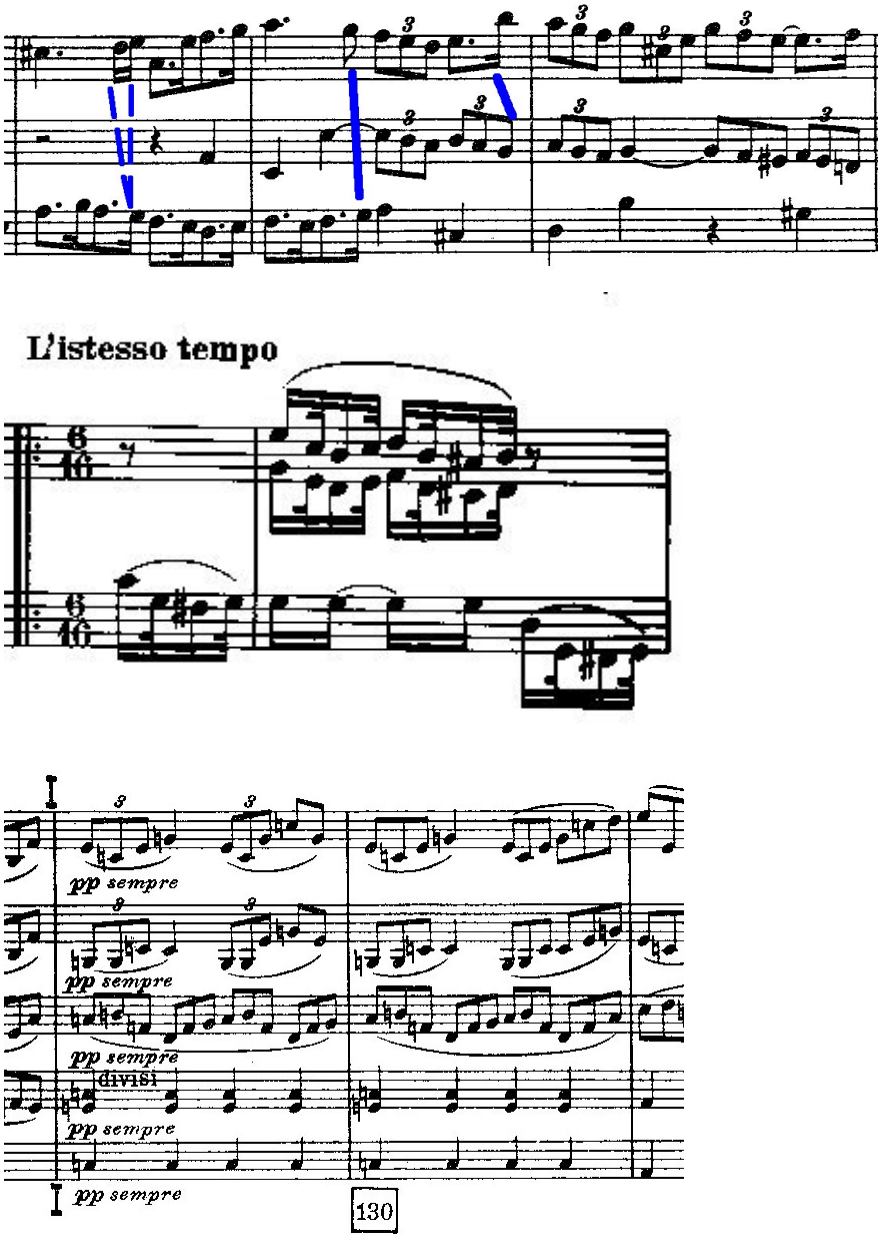


Figure 1: Authoritative Examples Violating **(RqArith)**

That a sixteenth note following a dotted eighth is played *synchronously* with the third eighth in a triplet is very frequently found in Baroque notation, because it perfectly satisfies **(RqErgo)**. So the last notes in the second measure in both upper voices are synchronous; as a consequence, also the second note in the top voice and the fourth in the lowest; the same interpretation is possible in the first measure, showing clearly that identical symbols have very different durations!

One can say, this practice violates **(RqArith)**, because when applying the simple method for interpreting the CWN duration values (as defined shortly, see Table 1), adding the duration values leads to arithmetic contradictions.

Otherwise, one could replace this simple method by a more complex one. But every possible solution would have to consider the context, and thus give up the valuable principle of locality ((**PropLocal)**, as defined below).

The second example in Figure 1 is from the second movement of Beethoven's op. 111. Here, triplet brackets are missing throughout, which is very sensible w.r.t. **(RqErgo)**. The same symbol, the sixteenths, in the upper and the lower system clearly have different durations. Again, any possible rule to fulfill **(RqArith)** would have to consider context information.

The third example (Bruckner, VI. Symphony) shows that triplet numbers are often left out when they are understood by the context.

In our approach we do not consider all these extended usages, but consider them as part of some heuristic "post-processing".

The metricSplit project only covers the first three requirements: **(RqArith)** is indispensable; **(RqErgo)** is in the center of interest; **(RqStyle)** is covered by an extensive collection of *style parameters* which modify the algorithm's operation.

1.4 Goals and Properties of Mathematical Models

The attempt of finding a precisely defined algorithm is always of (at least) dual use: First it is of course a prerequisite for automated processing. But furthermore it is also intended as a contribution to theoretic discussion, putting it on a more precisely defined basis. Esp., all rules and habits which are hard to formalize nearly always turn out to be critical also in other concerns.

For this purpose it is necessary to present the algorithm in a way understandable to the interested domain experts, here: musicians and musicologists.

The metricSplit algorithm comes with a collection of *style parameters*, which allow to adopt its behavior. It is therefore possible to apply it to corpora from history and categorize their way of notation by the parameter settings which will *reproduce* the same or similar results.

On the other hand, using the algorithm this way, or for type setting results of automated composition, will probably bring up the need for more or more precise parametrization, which includes a statement not only about metricSplit as such, but also about the nature of the task.

Furthermore, the operational structure of the algorithm can be considered a blueprint for the mental operations in the minds of musicians, over the centuries using and developing CWN. The analysis of any algorithm *as such* is always also a strict ethical requirement w.r.t. our responsibility for the development of human mind: Whenever automating a calculation process (here: engraving of sheet music) by delegating it to machines, the amount of daily practice in solving the particular sub-tasks decreases dramatically. This can lead to a loss in human problem solv-

ing potential and must be compensated by discussing the algorithms themselves thoroughly and in public.

This work is also meant as a contribution to the way of discussing music theory in general. We found it a fundamental lack of many approaches that there is no clean distinction between a notational *foreground* and a mental *middle-ground*, i.e. *notation* and *semantics*. It is well known in computing science, that mappings between these realms are often neither injective nor surjective. But many approaches take one for the other, without even mentioning the problem, treat e,g. sequences of duration symbols as rhythms and sequences of note heads with accidentals as pitch information. Of course this *may* be appropriate in some contexts, as an abbreviated way of speaking. But it is always a fundamental design error when talking about notation itself: Every notation system is, substantially, a collection of rules *mapping between* these two spheres, and to analyze the mappings the spheres must stay separated.

Last not least, every algorithm can give insight into the structure of the problem itself: Each property of the problem is possibly reflected by some property of the algorithm, as defined in the next section.

Mathematical modelling of a domain problem allows to localize, label and discuss properties of both in a more precise manner; properties of the algorithm may stand for properties of the problem and vice versa. The most important criteria in this concern are ...

- **(PropExpl)** — Explicitness:
  This property comes from the mathematical method itself and is unavoidable, but not common in humanities, where a large amount of "non-explicit pre-knowledge" can be a severe obstacle for discussion and communication. One of the main profits brought into humanities by mathematical modelling is explicitness, esp. when executing models by computers: All processing outcomes which appear "surprising" or "unintended" show that the specification of the algorithm is not yet fine enough, – rules for a computer must be given explicitly, or they do not exist.
- **(PropLoc)** — Locality vs. Context Sensitivity:
  Each part of an algorithm should clearly define the maximal subset of information it requires. By this, the corresponding part of the domain problem can be cooked down to its essentials.
- **(PropHeur)** vs. **(PropReg)** — Heuristics vs. Regularity:
  A similar dichotomy is between heuristics, which cover particular cases, against rule based processing, which covers all cases by general abstraction. This distinction is always an important criterion, which will e.g. show up in the CWN treatment of tuplets.
- **(PropComp)** — Compositionality:
  This property is assigned central importance in software engineering, but often neglected in modeling approaches. We state that whenever compositionality is violated, it indicates some idiosyncratic limitation, coming from particular ways of usage, historically established.

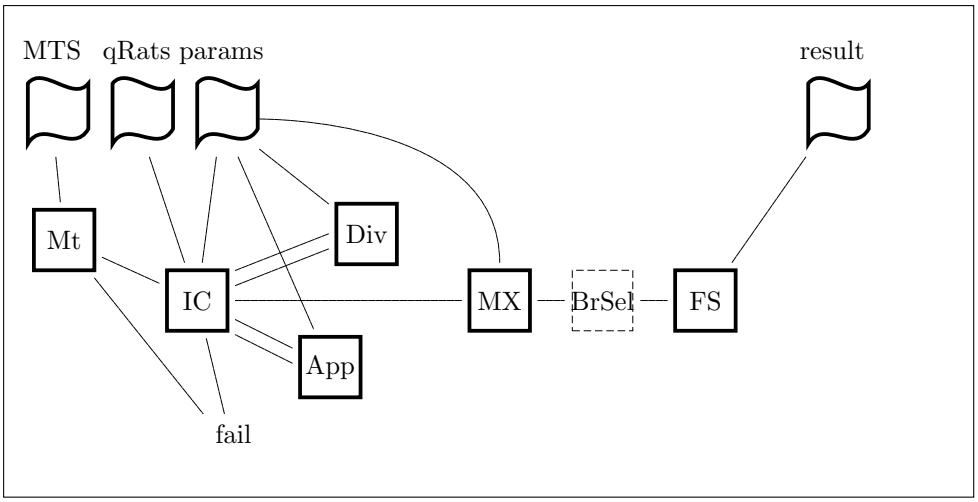


Figure 2: METRICSPLIT Processing Pipeline, Inputs and Results

2 The METRICSPLIT Algorithm

2.1 Processing Pipeline

The METRICSPLIT algorithm is realized as a *pipeline of transformation phases*, see Figure 2. It gets as its input ...

- a *metric tree specification (MTS)*, which represents the metric hierarchy,
- the rhythm to be notated as a sequence of times points, as positive rational numbers, qualified as audible or pause (sound or silence) *(qRats)*,
- and a whole zoo of style parameters which control the details of its operation *(params)*.

The output is a sequence of note duration symbols for each time point of the input. This includes the opening and closing of n-plet-brackets, duration symbols, prolongation dots and ties (see below for details). Additionally each note stem is accompanied by the description of its beams and beamlets, because this info also reflects the intended metric structure.[1]

2.2 CWN Duration Symbols and Rational Durations

The method for denotating rhythms in CWN is the result of a historic process over centuries. It is still ongoing, and its results have been coined by very different requirements from theory and practice.

The aspects relevant for our context can be described as follows:

Definition 2.1 (Notated Rhythm)**.** A *notated rhythm* is a sequence of *CWN duration symbols*. The sequential order of these symbols represent the flow of time. Groups of adjacent symbols represent *musical events* which shall be executed by some *musical voice*.

[1] In the current demo implementation `DemoMetric`, only the Musix-XML output uses this data; the conversion into LILYPOND source does *not* and leaves beaming to LILYPOND.

Each event may have arbitrary additional musical parameters. In most cases of CWN these include pitch or pitches, volume, articulation, etc. They are encoded in the notation by additional graphics or by the vertical position of the chosen duration symbols relative to the horizontal lines of the *staff*, or even by text. All these additional parameters are not in the scope of our problem. For each event we only have to know its duration, and whether it represents an *audible event* or a *pause*, i.e. *sound* or *silence*.

Definition 2.2 (CWN Duration Symbols)**.** The *CWN duration symbols* are

1. basic duration symbols
2. prolongation dots
3. ties
4. proportional brackets

Definition 2.3 (Notated Rhythm Grammar)**.** A notated rhythm in CWN can be seen as an instance of the non-terminal R of the following grammar:

$$\begin{array}{lcl} R & ::= & (S|E_P|P)+ \\ P & ::= & \mathtt{BR}(k, m, R) \\ S & ::= & E_S(\frown E_S)* \\ E_S & ::= & D_S(\mathsf{dot})* \\ E_P & ::= & D_P(\mathsf{dot})* \\ k, m & : & \mathbb{N}_{\geq 2} \end{array}$$

The semantic domain of a notated rhythm are *abstract CWN durations*. These are the durations on the level of notated music, before applying to them any translation into real-time values by execution tempo, expressive agogics, etc.

Definition 2.4 (Rational Durations)**.** A *rational duration* is a positive rational number interpreted as the duration of a musical event:

$$\mathbb{D} = \{r \in \mathbb{Q} \mid r > 0\}$$

In special cases, e.g. when measuring distances, the duration value of zero may be included:

$$\mathbb{D}_0 = \mathbb{D} \cup \{0\}$$

Definition 2.5 (Rational Rhythm)**.** A *rational rhythm* is a sequence of rational durations, meant to represent the temporal=rhythmic appearance of a sequence of sound and pause events.

Proposition 2.6 (Notational Semantics)**.** *The mapping from notated rhythms to rational rhythms is* always injective*. The mapping from rational rhythms to notated rhythms is* never injective*.*

These two lemmata form the basis and the challenge of this project, and of any attempt to solve the tasks **(TNoteSymbs)** and **(TSpont)**, as defined above. (Please note that the first lemma relies on the elimination of the irregularities from Figure 1, as discussed above. The second is shown most easy because we dropped the limit on flag counts (see below): Bi-division together with re-tie-ing is always possible and generates new notations of the same rhythm.)

The meaning of the nonterminals in Definition 2.3 and their mapping into the domain of rational durations is defined as follows:

P stands for a *Proportional Bracket*. CWN since its beginnings has supported mechanisms which split the duration of a note equidistantly among a "spontaneous" number of sub-intervals, others than those foreseen by the general metrum. The notational device is the *n-plet-bracket*, the most common form of which is the *triplet bracket*.

According to the above-mentioned tendency for abstraction, it developed from a local, spontaneous exception to a general purpose and compositional device. In contemporary notation such a bracket spans a sequence of adjacent notes r and is labeled with an expression "$k : m$". In the grammar, this is written as $\texttt{BR}(k, m, r)$. This means "throughout in r, perform k notes of duration x in the same time in which (without this bracket) m notes of the duration x would be performed". So this bracket puts the rational number m/k as a factor on all notated duration values of the note symbols spanned ($= r$). It is crucial that this factor is totally independent from the choice of x. This explicit labeling "k:m" is context-free and fully compositional, adheres to **(PropLocal)** and **(PropComp)**.[2]

In contemporary CWN, tuplet bracket in the notation front-end can overlap partially. The grammar rules above, and thus METRICSPLIT, support only proper nesting, i.e. of two overlapping brackets one must be completely contained in the other.

The two numbers k and m are in most practical cases relatively prime, but "6:4" is not seldom, for **(RqErgo)**. Considering **(RqArith)** only, it can be replaced by "3:2". It always holds that $m/k \neq 1$.

Furthermore, the sequence r of duration symbols contained in the expression $\texttt{BR}(k, m, r)$ in nearly all practical cases contains more than one element. (These two requirements are not expressed by the grammar.)

Definition 2.7 (Bracket Stack)**.** For a particular point in a Notated Rhythm, the *bracket stack* is the list of all n-plet-brackets which span this point. The product of the factors of these brackets is the *effective (n-plet) factor* at this point.

Definition 2.8 (Essential Bracket)**.** An *essential bracket (EB)* is labeled with $k : m$, and k is *not* a power of two(2).

Essential Brackets have this name because without them durations with that factor k in their denominator cannot be notated at all.

Definition 2.9 (Convenience Bracket)**.** A *convenience bracket (CB)* is labeled with $k : m$, and k *is* a power of two(2).

Convenience brackets have this name because they do not make more durations writable, but make their notation easier.

In the grammar of Definition 2.3, D_S and D_P stand for basic duration symbols from Table 1, which represent sound or silence, resp. Each of them represents a rational duration of the form 2^n, with $n \in \mathbb{Z}$ and $n \leq 1$. This takes the "Brevis" = 𝅜 for the longest duration symbol still in common use.

The table shows clearly the idiosyncratics of the historic development: Starting with the Brevis and repeatedly dividing by two, first the form of the note head is

[2] In contrast, the traditional way of labeling the bracket only with "k" must be seen as an abbreviation, where "m" is either fixed for a particular k, or must be reconstructed by analysis of all the spanned note symbols. In our approach, such a notation can only be introduced as an abbreviation, in a post-processing not discussed in this paper.

duration				*sound*	*pause*	
2	=	2^1	=			
1	=	2^0	=			
$\frac{1}{2}$	=	2^{-1}	=			
$\frac{1}{4}$	=	2^{-2}	=			
$\frac{1}{8}$	=	2^{-3}	=			(1 flag)
$\frac{1}{16}$	=	2^{-4}	=			(2 flags)
$\frac{1}{32}$	=	2^{-5}	=			(3 flags)
$\frac{1}{64}$	=	2^{-6}	=			(4 flags)
$\frac{1}{128}$	=	2^{-7}	=			(5 flags)
					(etc.)	

Table 1: Duration Symbols in CWN

significant, than the existence of the stem, then again the note head. Finally flags are added. Even more irregular are the pause symbols: Those with durations $\frac{1}{2}$ and $\frac{1}{1}$ are only distinguished by their position relative to the lines of the staff!

In CWN normally there is a limit on the number of flags; common practice does not descend below $1/128 =$  . This is ignored in our algorithm since it violates **(PropComp)**.

The duration of such a basic duration symbol in the context of a particular notated rhythm is its basic value, as taken from the table, multiplied by the effective n-plet factor of the enclosing brackets.

E_S and E_P stand for basic duration symbols plus arbitrary many *prolongation dots*. If the symbol represents the duration 2^n, the same symbol followed by k prolongation dots, with $k \in \mathbb{N}$ and $k \geq 0$ represents the rational duration $2^n * (2 - 2^{-k})$. (In a particular notation context, this again has to be multiplied with the effective n-plet factor.)

Definition 2.10 (Writeability)**.** For a particular bracket stack or effective n-plet factor, a given rational duration is *d-writable* iff it can be represented by a basic duration symbol from Table 1, i.e. as an instance of D_S or D_P from the grammar.

It is *e-writable* iff it can be represented by a basic duration symbol from Table 1 plus zero or more prolongation dots, i.e. as an instance of E_S or E_P from the grammar.

The *Tie symbol* ⁀ is a kind of addition operator: It joins two sound symbols which are instances of E_S, and their durations are added to get the duration of the intended event. (In practical notation this implies that the above-mentioned further "musical parameters" notated with the two symbols do not change significantly, or even are missing completely with the second symbol. The "legato symbol" looks exactly like the tie, but can connect notes with different pitch, etc. When only intensity changes, the differentiation between tie and legato can be a very subtle question, not in scope of this paper.)

A similar device does not exist for pauses; they are simply juxtaposed for addition; again a violation of **(PropComp)**.

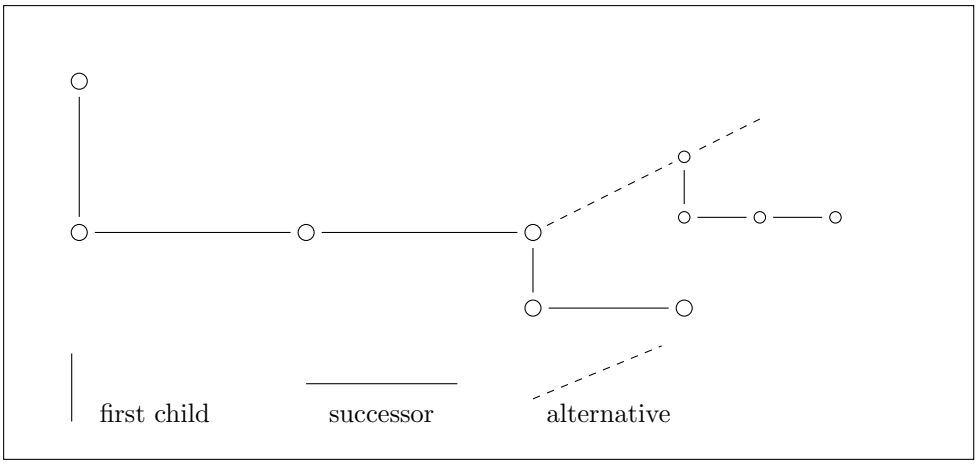


Figure 3: Shape of the Top of a Metric Tree (MT)

Definition 2.11 (Writeability-3)**.** For a particular bracket stack or effective n-plet Factor, a given rational duration is *writable* if it is e-writable, or is the sum of e-writable durations.

Every sequence of rational durations can be made writable: Find f as the Least Common Multiple of all odd factors of all denominators, and put a bracket $f : 2^x$ on the whole measure, i.e. as topmost construct of R, for some $x \geq 1$. Then all denominators of all durations are writable, and all enumerators can be constructed by addition, using the $\frown$ tie symbol.

2.3 Metric Tree (MT)

Definition 2.12 (Metric Tree)**.** In METRICSPLIT, each metrum is modelled as a *tree-formed graph* called *Metric Tree* (MT), with the following properties:

Each node of an MT represents a sub-interval of the duration of some measure which adheres to that metrum; the *root node* represents the complete time interval of the measure. The duration of this interval is also called the MT's duration.

A MT has vertices in three(3) dimensions, see Figure 3: Each node is related to a sequence of at least two(2) child nodes (vertical axis). The ordered sequence of child nodes of each node represents a sequence of adjacent sub-intervals of the parent node's interval, in temporal order (horizontal axis). Each such sub-interval has a *duration* > 0. As a consequence, each such sub-interval has a *start time* relative to the start of the interval represented by the parent node, namely the sum of the durations of all its predecessor siblings; and it has a start time relative to the start of the measure, namely that parent relative start time plus the measure relative start time of the parent node.[3] So the node has also relative *end times*, namely its duration plus the resp. start time. All these values are given as rational numbers from D_0. The sum of the durations of all child nodes is equal to the duration of the parent node.

[3] For shortness we say in the following "The node's interval" for "the interval represented by the node", "node's start time" for "the start time of the interval represented by the node", "the interval's successor" for "the interval of the successor node of the interval's node", etc.

Each node may have one node as its *alternative*, with the same start time, end time and duration, but a different internal structure (z-axis).

These alternatives are the main means for the user to control the generation of sensible notations for different rhythmic situations, as frequently required by conventional practice. Each alternative can be seen as reflecting a possible hierarchical structure in the musicians mind when performing: either (downward) dividing a beat into sub-events, or (upward) aggregating several beats into one phase. METRICSPLIT abstracts from all the different practices and models them all by MTs.

This approach puts no further restrictions on MTs: Child nodes may stand in arbitrary proportions to the parent node and among each other. A sequence like `1/4+1/5+1/6` (in the front-end syntax described in section 2.4) is perfectly valid for child nodes' durations. METRICSPLIT goes beyond classical CWN and supports metric structures needed in contemporary music, ethnology, for notating birds' songs, etc. The fundamental design philosophy is to stay in the well-behaving middle-ground of freely compositional mathematical structures which fulfill **(PropComp)**, like the rational durations, as long as possible.

CWN notation implies that every note symbol coming without further qualification can always be divided into two(2) symbols, each representing one half($\frac{1}{2}$) of the original interval's duration. In the notation front-end, this corresponds to the generic procedure of adding one more flag/beam to the representing note symbol's stem. This is generalized for MTs, and every node without any other specification implicitly has two(2) child nodes of the same duration. So every MT is at every node *infinite* in the direction of the child axis.

2.4 Input: Metric Tree Specifications (MTS); (Ph-Mt)

Definition 2.13 (Metric Tree Specifications)**.** A *Metric Tree Specification (MTS)* is a finite tree structure supplied by the user, meant to define an MT. It is denotated by an external representation which follows the nonterminal M from the following grammar:

$$\begin{array}{lcl} M & ::= & \underline{(}\ M\ \underline{)} \mid n \mid \mathtt{1} \mid n\ \underline{*}\ M \\ & \mid & p\ \underline{/}\ q \mid p\ \underline{/}\ q\ \underline{(}\ M\ \underline{)} \\ & \mid & M\ \underline{+}\ \ldots\ \underline{+}\ M \mid M\ \underline{|}\ \ldots\ \underline{|}\ M \\ p, q & : & N_{\geq 1} \\ n & : & N_{\geq 2} \end{array}$$

An MTS which is error-free specifies an (infinite) MT. The first processing phase of the METRICSPLIT algorithm is **(Ph-Mt)**, which calculates the MT and some additional auxiliary data structures.

First step: Construct a tree structure similar to the finite top part of an MT, according to the following rules:

- For the expression $r_1\ \underline{+}\ \ldots\ \underline{+}\ r_k$, construct a node the child nodes of which are described by the expressions r_1 to r_k.
- For a repetition expression $n\ \underline{*}\ r$ create one node with n child nodes, each of which has the structure defined by the expression r.
- For a plain number "$n > 1$" create one node, with a sequence of n child nodes with neither explicit inner structure nor explicit duration.[4]

[4] From these rules follows non-associativity: It is important to note that `2+3+3+3` $\neq$ `2+(3+3+3)` = `2+3*3`.

- For a plain number "`1`" create one node with neither explicit inner structure nor explicit duration.
- For the expression $a \underline{\ |\ } b$, set the node constructed for b as the alternative of that constructed for a.
- For the expression $p \underline{\ /\ } q \underline{\ (\ } r \underline{\ )\ }$, first construct the node defined by r. If this carries an (explicit) duration specification, the MTS is rejected as erroneous. Otherwise assign the rational number p/q as duration to that node.
- The expression $p \underline{\ /\ } q$ is an abbreviation for $p \underline{\ /\ } q \underline{\ (\ } 1 \underline{\ )\ }$.

Second Step: Propagate and check all duration values according to the following rules, until a fixpoint or a conflict is reached:

- If a parent node does not have a duration specification, but all of its child nodes have, the their sum is assigned to the parent.
- If a parent node has a duration, and all of its child nodes have, the sum of the latter *must be the same* as the former.
- If a parent node has a duration, and none of its child nodes, the duration of the parent is *equally distributed* among the child nodes.
- If a parent node has a duration, and so have all of its child nodes except one(1), then this one gets the difference (which must be positive) between the parent's duration and the sum of the children's durations.
- If a node has a duration, and its alternative has, both *must be the same.*
- If a node has a duration, but not its alternative, or vice versa, this duration is assigned to both nodes.

Whenever these rules lead to a contradiction or leave a node without duration, then the MTS does not specify an MT and is rejected as erroneous.

The front-end notation syntax and all these semantic rules have come from practical experiences. They follow **(RqErgo)** and **(RqHeur)**. They lead to utmost compact denotations and are expressive enough to notate all possible MTs.

The practical aspect implies that they could have been defined quite differently, as these examples demonstrate:

a)	`3/2 (3 3 3)`	OK
b)	`3/2 (1/2 3 3)`	erroneous
c)	`3/2 (1/2 1/2 3)`	OK

Example a) is OK because the top node's complete duration is equally distributed among the children; in c) the one(1) child node without an explicit duration gets the rest $= \frac{1}{2}$. But in the current implementation, such a rest is not distributed automatically among more than one child, see case b). In the design, too much robustness has been avoided, since compactness of notation and ease of error detection had to be balanced.

Third Step: The further processing cannot lead to errors, i.e. the node structure constructed so far is indeed the top part of an (infinite) MT. Now auxiliary static data is computed for the later application of the MTree structure:;

1) The start and end points of each node relative to the measure start are statically calculated.

2) The essential brackets (EB) are calculated. These are auxiliary pieces of information, preparing which nodes will require which kind of n-plet-bracket when later written to the CWN front-end representation.

The algorithm works as follows:

1. Visit the top-level node with a context factor $F = 1$.[5]
2. Visiting a node means: find all longest chains of adjacent subnodes with the same duration.
3. For each such chain set G to the denominator of this duration, divided by two as often as possible without rest.
4. If $G \neq F$, then store with the first node of the chain a new EB. This has $(G/F) : Z$ as its label, thus $\frac{Z}{G/F}$ as its factor, and the end point of the last node of the chain as its end point. Z is defined as that power of two which has the smallest distance to G/F. The middle between two powers of two is mapped upward.
5. Visit all nodes in the chain, with G as the new context factor.

Similar to the front-end notation, the *EB stack* of a particular node is the sequence of all EBs which cover the time interval of this node; the *effective n-plet factor* is the product of the factors of all these EBs.

This calculation can be made complete "on stock", because the automatic extensions of an MT, i.e. the lower leaves added later on demand, only introduce the factor 2 and thus never need additional EBs.

3) For each node, the symbols from E_S and E_P are calculated on stock (= basic symbol, including the number of flags/beams, plus number of prolongation dots) which represent the duration of the node under the node's effective n-plet factor. If they exist, these symbols will later be used for output; but they may be = `null` , if the duration is not e-writable, either because of complicated enumerators, or when exceeding the maximum of allowed prolongation dots given by **Param:**`max_dots_positive` and **Param:**`max_dots_for_pauses`[6]

4) The standard beam and beamlet configuration is calculated on stock for every stem. This describes the default beaming which will be printed if every node is represented by a sound event. It will possibly be modified before printing when pauses are inserted. This calculation is performed initially once, when instantiating an MTree. It is resumed later, whenever the MT will be extended, for the newly created subnodes. It works as follows:

1. Independently of its duration, the top node gets a StemEnd description with zero beams and beamlets to either side. (This is only relevant for an MT with a total duration of $\frac{1}{8}$ or shorter.) Then its subnodes are visited.
2. Visiting means: The very first subnode inherits the number of left beams and left beamlets from its parent.
   The very last subnode inherits the number of right beams and right beamlets from its parent.
3. For every pair of adjacent inner subnodes, the right beams of the left (=earlier) node and the left beams of the right (=later) node are set to the minimum of the resp. flag counts, as calculated in the preceding step 3).
4. If the right node's duration requires more flags than it has beams, additional beamlets to the left are added.
5. If the left node's duration requires more flags than it has beams, additional beamlets to the right are added, only if it is the very first subnode, or the style parameter

[5] The current implementation `DemoMetric` does not support an odd factor in the denominator of the duration of the topmost node, i.e. of the whole measure described by the MTS.

[6] Dot limits are not tested in the current implementation.

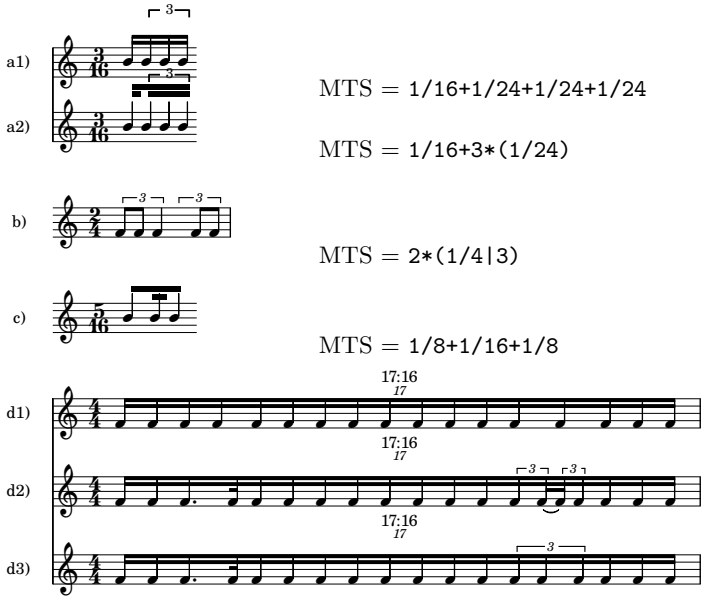


Figure 4: Examples of Beams and Brackets

Param:`additional_short_towards_complement` is set to `true`. The latter case yields a notation style which can be sensible for very particular notation situations in avantgarde or ornithology, see Figure 4, line c).

6. When the data of a node has been completed, its subnodes are visited recursively.

This particular beaming strategy directly reflects the node structure: Notes with different durations like $\frac{1}{16}$ and $\frac{1}{24}$ will not be separated by gaps in the beam, if the corresponding nodes are direct siblings as in `1/16+1/24+1/24+1/24`, – a clumsy specification anyhow, see Figure 4, line a1). The more sensible MTS `1/16+3*(1/24)` is of course rendered as shown in line a2).

Figure 5 shows a typical control visualization of an MT data structure, as created so far: The hierarchy of nodes is visualized by indentation; the horizontal axis of child nodes is shown by numbering; the roots of alternatives carry the same number; each node has a duration, together with an exclamation mark if this duration has come from the MTS; start and end time are printed as "`[_->_]`"; finally shown are all EBs starting with this node, with proportional factor and end time (starting with "`(br=`"; for convenient later processing, the end of an EB is encoded as a measure relative time point).

With `DemoMetric`, an MTS expression (according to the grammar from Definition 2.13) must be entered to specify the MT for further processing. The first page of the GUI shows on the left side the resulting MTS, and on the right side its evolving state when further requests for rhythmic rendering extend its definition automatically, as described below.

2.5 Input Data qRat

Definition 2.14 (Input Rhythm)**.** For any $s : \mathbb{N}_{\geq 2}$ and any MT with a duration d, the data type qRat defines the legal input data to represent the rhythm to be

```
 MTS = 1/2(2*(2|3))   ===>

(0)--- !1/2    [0 -> 1/2]
    (0)---  1/4    [0 -> 1/4]
        (0)---  1/8    [0 -> 1/8]
        (1)---  1/8    [1/8 -> 1/4]
    (0)ALT  1/4    [0 -> 1/4]
        (0)---  1/12    [0 -> 1/12]       (br=[2/3 --> 1/4]
        (1)---  1/12    [1/12 -> 1/6]
        (2)---  1/12    [1/6 -> 1/4]
    (1)---  1/4    [1/4 -> 1/2]
        (0)---  1/8    [1/4 -> 3/8]
        (1)---  1/8    [3/8 -> 1/2]
    (1)ALT  1/4    [1/4 -> 1/2]
        (0)---  1/12    [1/4 -> 1/3]       (br=[2/3 --> 1/2]
        (1)---  1/12    [1/3 -> 5/12]
        (2)---  1/12    [5/12 -> 1/2]
```

Figure 5: A Typical Control Dump of the Top Part of a Metric Tree

rendered. It is defined as a sequence of "qualified rationals", namely

$$\mathsf{qRat} = \{0..s\} \to (D_0 \times \{\ \underline{\texttt{true}}\ ,\ \underline{\texttt{false}}\ \})$$

with the following additional properties for each instance $r \in \mathsf{qRat}$:

- $u < w \iff r(u) < r(w)$, i.e., the sequence of time points is strictly increasing.
- $r(0) = (\frac{0}{0}, \_)$, and
- $r(s) = (d, \_)$, i.e., the whole interval of the measure is covered.

This sequence has the interpretation, that at every time point (except the very last) a musical event starts, which shall be rendered in CWN notation, and which extends to the next time point in the list. The "qualifying" Boolean value `true` indicates that this event is audible, and not a pause, i.e. the meaning of the Boolean value is "sound, not silence".

The `DemoMetric` demo application supports an alternative input format, namely a sequence of "qualified durations", meaning the durations of events, the first of which starts at the beginning of the measure (time point $\frac{0}{0}$). Inputs in both formats are first *normalized* according to the principles listed above, and shown in both formats in the dedicated input fields of the GUI. (When calling the API directly, under program control, of course the caller is responsible for fulfilling the interface contract.)

2.6 (Ph-IC): Find Initial Coverage

In the first step the algorithm searches for the *initial coverage* (IC) for each event. This is a finite list of nodes from the MT which has minimal length and the intervals of the nodes adjacently cover the interval of the event exactly.

If no such IC exists, this is due to a time point which has a denominator not contained in the divisions foreseen in the MT. Then the user of METRICSPLIT can choose between three ways of reaction:

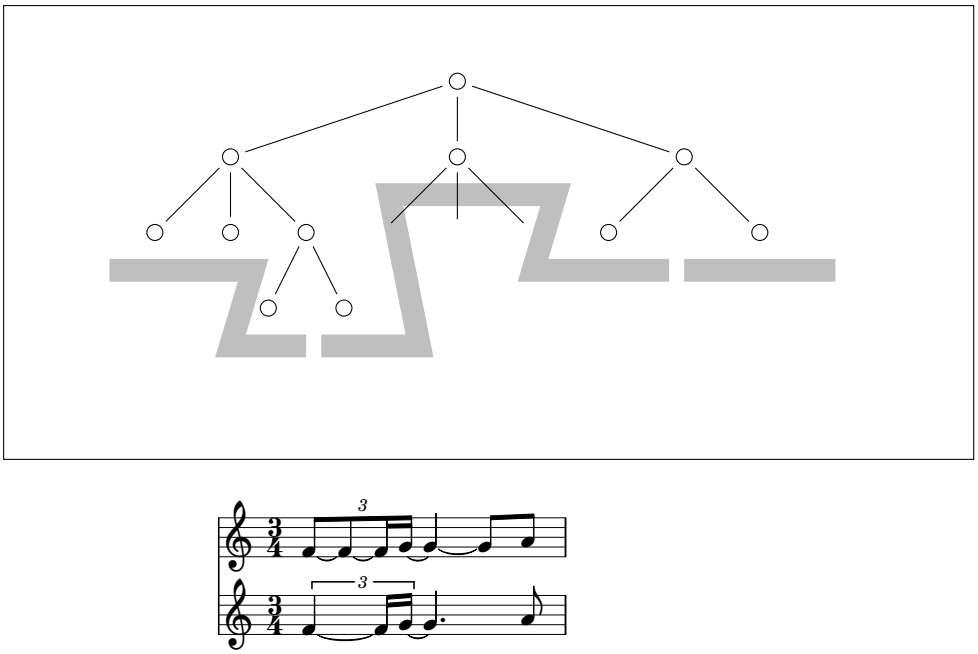


Figure 6: Example for an Initial Coverage (Ic, Icn) And Result of **(Ph-MX)**

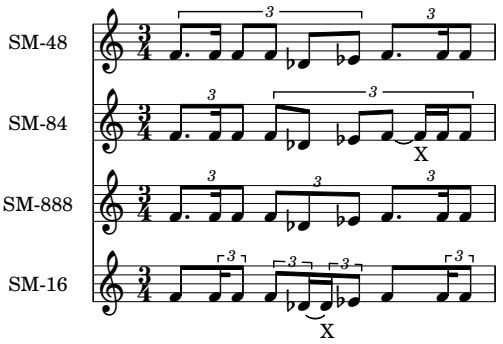
- The application fails.
- Phase **(Ph-Div)** is called. This synthesizes the required new sub-divisions, which provide the missing division factors exactly.
- Phase **(Ph-App)** is called. This creates an approximation by repeated equidistant division.

When the first alternative is selected, the METRICSPLIT algorithm is a *partial function* from MTS, rhythmic input and style parameters into notated rhythms, as defined above. In the other cases it is *total* (notwithstanding the cases of erroneous input, as described above).

The nodes coming out of **(Ph-IC)** can always be written as CWN duration symbols, using the required EBs. In most cases, but not always, the durations are e-writable or even d-writable. For every event, rowing the corresponding notation symbols in the sequential order of the nodes of the IC and joining them by tie symbols yields a first possible notation, the *initial coverage notation (ICN)*. This is "adequate" in the sense of **(RqErgo)**, as (a) every note symbol appears at the start time of a node of the MT, and (b) vice versa every node which is covered by the event, but not its parent, is visible in the notation. Nevertheless, the ICN is hardly ever used in practice for notation, but further transformed in **(Ph-MX)**. Figure 6 shows symbolically an IC, the corresponding ICN and how **(Ph-MX)** will simplify it.

2.7 Usage of Explicit Alternatives; Algorithm (Ph-IC)

The alternatives $a \underline{|} b$ in the MTS are the means for the user to control the appearance of sporadic and local changes to the metrum. A frequent example, found even in very simple compositions, are triplets of eighths appearing in a 4/4 metrum.



MT=3/4	2/4(2|3)+1/4	3*1/4(2|3)
X	X	
X		
X	X	X
X		

Figure 7: Different Ways of Notating a 9/8 Triplet Rhythm Into a 3/4 Metrum

If such a division by three is not foreseen in the MTS, then an automatically synthesized alternative will always satisfy **(RqArith)**, but in general not **(RqErgo)**, **(RqInt)** and **(RqStyle)**. This is because a missing factor in the denominator can be synthesized on any level of the MT with equal rights.

Trying to find objective criteria for the selection is ongoing research and has not yet brought satisfying results. Only the notes marked with "X" would fail an objective criterion "minimal symbol cont" and thus exclude SM-84 and SM-16.

Further candidates for an objective criterion are "homogeneity" and "minimal node count", which would automatically prefer SM-888. But we have not yet succeeded in a general automation with satisfying results in practice.

In METRICSPLIT, the user can easily control the selected way of notating by supplying different MTSs:
"`3/4`" may yield anything,
"`2/4(2|3)+1/4`" will yield only SM-48,
and "`3*1/4(2|3)`" will yield only SM-888.

The style parameter **Param:**`best_fit_not_first_fit` decides the strategy: With "first fit" the first alternative found on the on z-axis is taken, which allows to represent the rhythm without adding "synthetic" nodes (as described in the next section); "best fit" takes that alternative which needs least nodes and least additional synthetic nodes, after trying all.[7]

The dual case to these explicitly controlled n-plets are spontaneous denominators which appear totally unforeseen, e.g. coming from automated composition or analysis. For these, **(Ph-Div)** automatically adds new alternatives to the MT as necessary. This happens only below the leaf nodes of the MTS; all automatic divisions are lower in the metric hierarchy than those explicitly defined by the user.

This phase is the most complex in METRICSPLIT, and the results are in most cases satisfying w.r.t. **(RqErgo)** and **(RqCons)**. The may fail **(RqStyle)**; in this case the algorithm can be re-applied to the same event data, with MTS and style parameters modified accordingly. The phases **(Ph-IC)**, **(Ph-Div)** and **(Ph-App)** are conceptually separate, but implemented in an interwoven style, switching to and fro as required.

The different steps are implemented as co-routines, working on a `Cover` object as

[7] In `DemoMetric`, first fit is not yet implemented.

their context. This has as mutable fields

- the current node
- the current event
- a map from the input events to one list of MTree nodes each. These are called *node lists* of the events and make up the true result of **(Ph-IC)**.
- some quality values, which make this `Cover` comparable to its alternatives.
- a map from the nodes of the MT to new synthesized alternative nodes. These will be inserted into the MT when the solution, represented by this context, is finally accepted.

The start times of all events are given globally, and all events are identified by their index. The end time of event n is the time point at index $n + 1$.

The main matching routine is simple and is called for one particular MTS node N as the *current node* and one particular time point index i as the *current event*. The process is started by calling it with $i = 0$ and $N =$ the root of the MT:

- (MATCH:)
  If $N.endTime \leq T_{i+1}$, then add N to the node list of the event and return.
  If $N.endTime = T_{i+1}$, then additionally advance the context value "current event".

- Otherwise, only a part of the duration of N belongs to the event i, and a finer node structure must be found:
  If N has an alternative, then create a new copy c_1 of the current `Cover` context. With c_1 as context, test the subnodes of N, as defined in (VISITSUBS).
  If this yields a result which does not need synthetic nodes, and **Param:**`best_fit_not_first_fit` = `false` , then add the results (node lists, quality values, map of new alternatives etc.) of c_1 to the original context, restore the original context and return.
- Otherwise create a new context c_2 and visit the alternative as described in (VISITSUBS). After returning, take the cover with the smaller number of synthetic nodes, or, if equal, the smaller number of all nodes, as result. (These values are cached in the quality values of the contexts.) Add the values of chosen result to the original context, restore it and return.
- If N does not have an alternative, then (VISITSUBS) can be called with the original context immediately.
- (VISITSUBS:)
  The test of the alternatives works by simply calling (MATCH) for the subnodes, one after the other, passing the context from one call to the next.

2.8 (Ph-Div): Synthesis of New Alternatives

All these steps described so far do only consider explicit nodes from the MTS. Synthetic and implicit nodes are "invisible". If (under this view) a node subject to (VISITSUBS) does *not have subnodes*, then a leave node of the MTS has been reached, i.e. a point where the author of the MTS did not specify the finer structure. In this case this finer structure must be synthesized automatically, and a kind of *spectral analysis* is started. All time points which fall into the duration of this node are collected and their denominators are analyzed.

This implies *a fundamental change in paradigm*: preceding phases, including the matching of nodes described so far in **(Ph-IC)**, follow **(PropLoc)**: they always consider locally only one (1) particular event and the corresponding part of the MT. Here more than one event (> 1) are considered, making the algorithm somehow "context sensitive".

The spectral analysis enumerates the prime factors of the denominators of the distances of the events relative to the start time of the node.

If these are only powers of two, then the implicit bi-division, which is ubiquitous in CWN, is sufficient to realize the necessary finer structure: A modified version (MATCH^B) of the (MATCH) algorithm is called, which does not follow the alternative axis, but which creates two equidistant subnodes with every call corresponding to (VISITSUBS), if this has not yet happened.

But if there are odd prime factors, subnodes must be synthesized which are not implicitly defined in CWN. For this, a new node N_A is generated and entered into the MT as an alternative to the existing node. N_A gets the required number of subnodes, plus the necessary EB to realize the duration factor, plus possibly further odd subnode structures on deeper levels of nesting. (N_A will be stored in the current context and added to the global MT nor before this alternative has indeed been finally chosen as the result of the overall process.)

Normally there will be only one such missing odd factor, mostly 3, 5 or 7. Combinations like 3*5 etc. are also possible, up to a CHOPINesque division by 23, which includes a triplet or 5-plet somewhere in its middle.

With only one such factor, N_A is simply constructed with the given number of subnodes and the required EB. Then (MATCH^B) can be called, because under this EB only bi-divisions will appear on all finer division levels.

But with more than one factor, a *stacking plan* L of the factors must be determined. This is the critical task w.r.t. **(RqErgo)**, and a central achievement of METRICSPLIT.

First L is constructed as an ordered list of factors, then the node N_A is created as an alternative, then (MATCH^D), a further modified version (MATCH), is executed. Like (MATCH^B) it does not follow alternatives. When in (VISITSUBS) subnodes are missing, a short analysis of all time points falling into the node's interval is executed, whether there are odd factors in the denominators (= a simple prefix of the spectral analysis described above). If not, (MATCH^B) is called for binary division. If yes, the number at the head of L is used for dividing the current node, and (MATCH^D) is called with the tail of L. If L has shrunk to empty, then the full spectral analysis is re-executed, as described above, and (MATCH^D) re-entered with a newly calculated L.

The spectral analysis and the construction of the stacking plan has these input values:

$D_M : \mathbb{D}$ // = *duration of the whole node A*
$T_M : \mathbb{D}_0$ // = *start of that node*
$k : \mathbb{N}$ // = *number of time points to process*
$0 \leq n < k$ // = *running index of time points which fall in A*
$T_n : \mathbb{D}_0$ // = *end of event number (n-1) = time point number n*

Now the time points are normalized: first to fractions of the node interval's

duration, then to integral multiples of the largest common fraction (which is the inverse of the least common multiple of all denominators). All further calculations are made with these integer values:

$$
\begin{array}{ll}
f_n = (T_n - T_M)/D_M & // = \textit{start point as a fraction of the whole duration} \\
F : \mathbb{N} & // = \textit{least common multiple of the denominators of all } f_n. \\
I_n : \mathbb{N} & // = \textit{start point as multiple of common fraction } \frac{1}{F}. \\
I_n = f_n * F &
\end{array}
$$

Let *odd prime factors* be modelled by a partial map from bases to exponents:

$$\mathbb{P} = \mathbb{N}_{\geq 3} \nrightarrow \mathbb{N}_{\geq 1}$$

The empty map is called 1 and represent the maximal odd factor 1 (i.e. the absence of odd prime factors). The following operations are defined on $\mathbb{P}$, by distributing over the exponents of the same bases; a basis b missing in one of the maps is treated as $b \mapsto 0$:

$$
\begin{array}{ll}
\{3 \mapsto a_3, 5 \mapsto a_5, \ldots\} + \{3 \mapsto b_3, 5 \mapsto b_5, \ldots\} & = \{3 \mapsto (a_3 + b_3), 5 \mapsto (a_5 + b_5), \ldots\} \\
\{3 \mapsto a_3, 5 \mapsto a_5, \ldots\} - \{3 \mapsto b_3, 5 \mapsto b_5, \ldots\} & = \{3 \mapsto (a_3 + b_3), 5 \mapsto (a_5 - b_5), \ldots\} \\
\mathsf{min}(\{3 \mapsto a_3, 5 \mapsto a_5, \ldots\}, \{3 \mapsto b_3, 5 \mapsto b_5, \ldots\}) & = \{3 \mapsto \mathsf{min}(a_3, b_3), 5 \mapsto \mathsf{min}(a_5, b_5), \ldots\} \\
\mathsf{max}(\{3 \mapsto a_3, 5 \mapsto a_5, \ldots\}, \{3 \mapsto b_3, 5 \mapsto b_5, \ldots\}) & = \{3 \mapsto \mathsf{max}(a_3, b_3), 5 \mapsto \mathsf{max}(a_5, b_5), \ldots\}
\end{array}
$$

Let P_F be the odd prime factors of F and let P_n be the odd prime factors of the denominator of f_n.

The most simple case is that P_F is empty; then no odd divisors are in T_n, and this procedure would not have been called.

The next simple case is that all f_n have the odd prime factors 1 or P_F, i.e. there is only one single odd prime factor for all time points. In this case P_F is translated into a stacking plan immediately, as described below (=case X1).

Let G be the greatest common divisor of all P_n:

$$G = \mathsf{min}(P_0, P_1, \ldots P_k)$$

If $G > 1$, then this factor is common in all factors, and the method is restarted with a stacking plan $L = \langle G \rangle$ (= case X2). As a consequence, one(1) EB spanning the whole duration D_M will be synthesized and G subnodes with the duration D_M/G will be inserted as subnodes of N_A. These subnodes will be visited separately. Since $G \neq P_F$, this will lead to further EBs, each spanning one subnode or one indirect subsub..node.

(This has the effect that e.g. the factor "15" is realized by one single bracket, if there is no event which requires *only* a "3" or a "5". A factor "45" will be realized by an additional bracket, created later, when visiting the subnode. In most cases, these results are sensible w.r.t. **(RqErgo)**.)

If $G = 1$, then there are totally disjoint prime factors, and we have to find out how to arrange them. Let p_{max} be the P_n with the largest numeric value (=product of all factors). We calculate a *spectral analysis triangle*

$$A : \{2 \ldots p_{max}\} \to (\mathbb{N} \nrightarrow \mathbb{P})$$

We consider every $f \in \{2 \ldots p_{max}\}$ a possible top-level divisor, i.e. the number of equidistant sections, into which the duration of the top-level node is cut. (This

either requires an EB, or f is a power of 2). P_f are the odd prime factors of f. Then we calculate the prime factors which are still required in the different sections. The complexity of these factors is translated into a quality measure, and one of the f with the best quality will be taken.

In detail, this works as follows:

- For each $f \in \{2 \dots p_{max}\}$:
  For each $x \in \{0 \dots f-1\}$:
  Initialize $A(f)(x) = 1$ (i.e. set it to an empty prime factor map).
- For each combination of $f \in \{2 \dots p_{max}\}$ and $0 \leq n < k$:
  Find the segment number of the time point n as $s = I_n \mathsf{div} (F/f)$.
  Set the odd prime factors $P = f_n - P_f$ (i.e. those which are still unrealized after the division by f).
  Set the accumulator $A(f)(s) := \mathsf{max}(A(f)(s), P)$.

Now each divisor gets a *quality value* Q_f. Let f_F be the number of factors in P_F (= the sum of all exponents in the odd prime factor map of the LCM of all denominators).

For each $f \in \{2 \dots p_{max}\}$ do:

- Initialize $Q_f := 0$.
- For each $s \in \{0 \dots f-1\}$ do:
  Let g be the numbers of factors in $A(f)(s)$ (= the sum of all exponents).
  If f is odd, not even:
  - If $g > f_F$, then the number of factors increased, because f is not a factor in F; give infinite penalty: $Q_f := Q_f - 10000$
  - If the number of factors $g > 2$, give penalty: $Q_f := Q_f - 18$
  - If $g = 2$, give penalty: $Q_f := Q_f - 6$
  - If $g = 1$, give penalty: $Q_f := Q_f - 3$

  Otherwise, if f is even, the penalties are different, because bi-division shall only be applied if it really brings some benefit.
  - If $g \geq f_F$, then the number of factors has not decreased; give large penalty: $Q_f := Q_f - 1000$
  - If the number of factors $g > 1$, give penalty: $Q_f := Q_f - 4$
  - If $g = 1$, give penalty: $Q_f := Q_f - 1$
- Find the highest value q_M in the range of $Q_\_$.
- Find all best divisors $D = \{x : \mathbb{N} \mid Q_x = q_M\}$.
- In case D contains a power of two(2), then the largest such from that set is of taken; this limits the necessary brackets to span the minimal possible durations, see line b2) in Figure 8.
  Otherwise:
  If **Param:**`finer_division_down_not_up`, then take the smallest member in D as next divisor d, otherwise the largest. Set the stacking plan $L = \langle d \rangle$.

If there is maximally only one and the same odd factor f in all input time points, marked as (case X1) above, then this factor can be prime or compound. If its compound, its external appearance can vary and is controlled by style parameters:

- If **Param:**`parameters_divide.prefer_one_bracket` is true, than only one EB is used.

- If $f = 15$ and **Param:**`prefer_one_bracket_15_to_16` is true, then only one EB is used.
- If $f = 9$ and **Param:**`prefer_one_bracket_9_to_8` is true, then only one EB is used.
  These cases come from the special role these proportion have in particular styles and practices; thus they are not generic but follow **(PropHeur)**.
  In all these cases the stacking plan is $L = \langle f \rangle$.
  The resulting EB will have the label "$Z : f$", with Z being the "nearest power of two", as defined for "on stock" EBs in section 4 on page 18.
- Otherwise, if **Param:**`finer_division_down_not_up` is true, then the complete list of prime factors of f appear in ascending order in L, otherwise in descending order.[8]

This strategy could also be applied in case $G > 1$, which is (case X2) from above. But this is currently not implemented.

A totally different strategy can be selected by **Param:**`recursive_separation` = `true` . This applies the bi-division, which is implicitly defined for every node in CWN, until different prime factors fall below different MT nodes. This means for the outer representation, that EBs are never nested, but only juxtaposed.

This is implemented in a most easy way: If there are odd factors, and $G > 1$, then call the procedure recursively with a stacking plan of $\langle 2 \rangle$. This will lead to bi-division of the current node, until all prime factors are separated.

An extreme way to use METRICSPLIT is to supply the utmost simple MTS "`1/1`", defining no internal structure at all. Then *all* denominators $\neq 2^n$ will be realized by synthetic divisions from this phase. This corresponds to a purely "technical" usage of CWN rhythm notation, to a totally *non-expressive metrum*.

Figure 8 shows some complex results:

Lines a1) and a2) shows a rhythm for which stacking plans a1) $\langle 3, 5 \rangle$ and a2) $\langle 5, 3 \rangle$ have the same quality. All time points have the odd prime factors 15; the parameter **Param:**`finer_division_down_not_up` decides.

In Line a3) the finer division 5 is put atop the 3, against this parameter, because an additional event has the factor "5 only", which forces the common divisor to the top.

Lines b1) and b2) show that repeated bi-division succeeds in separating different odd factors. In Line b2) the finer division is taken, as described above EBs spanning two half notes (=the complete measure) would have the same value from A as the selected quarter notes.

Line c) shows that a "3" is taken for top-level, since there exists on time point with odd factors "3 only". Replacing 1/3 by 2/9 *and* setting **Param:**`prefer_one_bracket_9_to_8` will result in one single EB.

Line d) shows that a "5" is taken for top-level, because it is the common divisor.

Line e1) shows that binary separation as in b) is not done automatically if the power of two is higher than the highest odd factor. So the both factors are indeed

[8] "Complete" means: Every basis with exponent q appears q times.

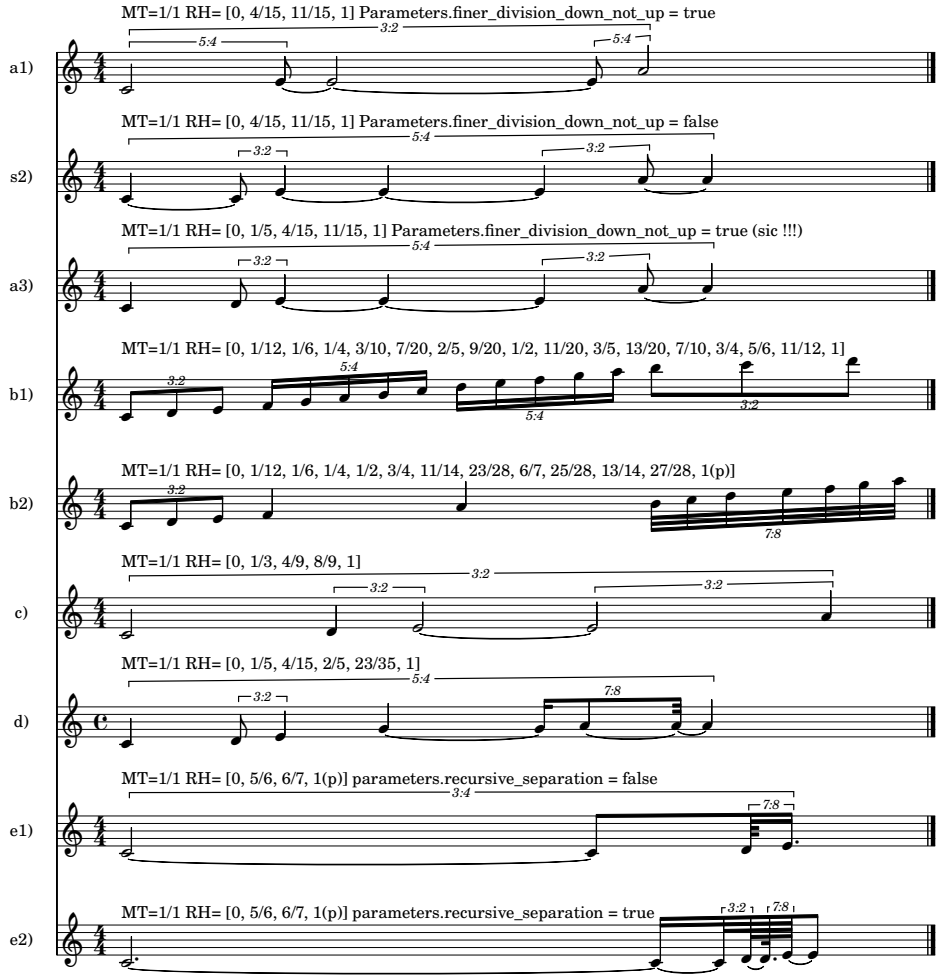


Figure 8: Examples of Synthesized Complex Divisions

independent, but are nevertheless printed in a nested way = horizontally combined. So the factor "6" is compensated by dotting the very last symbol, because it is superfluous in the very last duration.

Line e2) shows that activating **Param:**`recursive_separation=true` replaces unnecessary horizontal nesting by juxtaposition.

Line e1) shows additionally, that restricting the EP to the second half of the measure could be sensible w.r.t. **(RqErgo)**. But for this, we have not found any easy rules yet.

2.9 (Ph-App): Approximation by Repeated Division

This phase is an alternative to **(Ph-Div)** to make the operation of metricSplit a *total* function: It does not introduce new exact divisions, but replaces unsupported denominators by approximations. It divides durations repeatedly by one particular, fixed and simple factor, until the difference between original and replacement falls below a certain threshold. There are style parameters for the factor and for upper and lower thresholds.

With factor = 2, this procedure can be used as an explicit variant of what *digital sequencers* tend to do under the hood, namely rounding to binary divisions. **(Ph-App)** is not discussed in this paper, and currently only partially implemented in the demo application `DemoMetric`.

According to the principle **(PropComp)**, the decision between failing or applying **(Ph-Div)** or **(Ph-App)** does in no way affect the other phases.

2.10 (Ph-MX): Source Patterns for the Merging Transformations

(Ph-MX) gets an IC for every event as its input. This is (as described above) a map from each event to a set of nodes from the MT. These nodes could be notated (in the temporal order defined by the MT) and joined by tie symbols, which gives the ICN.

In historical as well as contemporary practice, the ICN is hardly ever used directly. Instead, further *merging transformations (MXs)* are applied, which can replace a sequence of note symbols from the ICN by one single note symbol, – see Figure 6 for a simple example. MXs reflect that metrum serves as a means of concrete practical communication and coordination; it comes with mental patterns and methods for its real-time execution; it implies rhythmic feelings and stress patterns. The patterns which are applied in the MXs originate from these purposes, they are unnecessary for **(RqArith)** but indispensable for **(RqErgo)** and highly varying with **(RqStyle)**.

metricSplit follows **(PropComp)**, as this phase operates identically on any IC, coming from any pregoing phase: Whether an explicit variant, foreseen in the MTS has been used, or an implicit bi-division has been applied, or a spontaneously created new division (see line d2 of Figure 4 on page 19), or a binary approximation, – the rules and the outcome of this phase are always the same.

The output of this phase is the Boolean predicate for each node from the IC, whether it shall be printed by a duration symbol, or its duration added to its last printed predecessor. The MXs are pattern driven transformations. For every single event (=sequence of nodes from MT), the applicabilities of the transformations are tested separately: The algorithm follows **(PropLocal)** again.

It is important for the design of the algorithm, and an aspect of **(PropComp)**, that the left side patterns of the transformations have incompatible shape, see

Figures 9 and 10 . Therefore they can be tested independently; the only possible overlap are the "knees" in Figure 10.

Closed transformations impose constraints on both the start and the end node of an event and replace its whole node sequence. As a consequence, their result (e.g. a syncopated note) is never tied to a further note symbol. This limitation rises naturally from their origin in performance pragmatics; it is related to **(RqErgo)** more than to **(RqStyle)**.

Open transformations only impose constraints on few relations between the nodes of sub-sequences of an event and can be tied to the results of other transformations, if further conditions are fulfilled. Therefore they can *compete* about nodes and their application is more complicated and includes conflict resolution.

An important case is *hierarchical combination* of MXs, e.g. a dotting "inside" a syncope, This is *currently not supported* by METRICSPLIT.

The closed transformations are ...

- *syncope merge* (MX-Y) — a suffix of the child sequence of a parent node, plus a prefix of the child nodes of its successor can be represented by one(1) single note symbol, if this concatenated sequence and both parents all have the same duration. Only if the parents are divided by two(2), then MX-Y is applicable also to cousins or great-cousins, as long as they are adjacent in time (see examples in Figure 9). That this not possible with other factors than two comes from **(RqErgo)** and **(RqStyle)**, and introduces some **(PropHeur)** into the algorithm.
- *hemiola merge* (MX-H) is similar to MX-Y, but involves parents divided by three of length $3 * u$ and an overlapping event of length $2 * u$.

For every event the closed patterns are tested first. If applicable, **(Ph-MX)** is complete for this event, and only its first node will be visible. The Boolean style parameters **Param:**`syncope_2_3` and **Param:**`syncope_3_3` enable the two possible starting points of syncopes of length three(3); **Param:**`syncope_longer_4` enables longer syncopes, e.g. of five or seven sub-units; hemiolas are enabled by **Param:**`hemiolas`.

Parameter **Param:**`int max_level_syncope_2` limits the number of levels crossed, in case of two-based syncopes. See Figure 9 for a real-world example: The composer and the editor of the critical edition indeed have chosen variant "P3". This may be acceptable, but the same setting with different contents, see "P3e", hardly fulfills **(RqErgo)**. It is valuable to preserve **(PropLocal)** and not to consider context information. With METRICSPLIT this is possible: Setting **Param:**`max_level_syncope_2=2` yields satisfying results in both situations, as line "P2" and "P2e" show.

Syncopes are often not applied to pauses, only to sounds. This fact belongs to the **(PropErgo)** and **(PropStyle)** realms. The current implementation of METRICSPLIT does not apply them to pauses at all, but a finer granularity of control could be desirable; the corresponding extension of the style parameters set would be easy.

The open transformations are ...

- *siblings merge* (MX-S) — more than one adjacent nodes with the same parent and the same duration can possibly be represented by a single note symbol.
- *dotted merge* (MX-D) — one node and the left child of its right sibling can be represented by one single note symbol, if the right sibling is equidistantly divided by two and both siblings have the same duration. This can be repeated recursively (i.e., the left child of the right sibling, *plus* the left child of the right

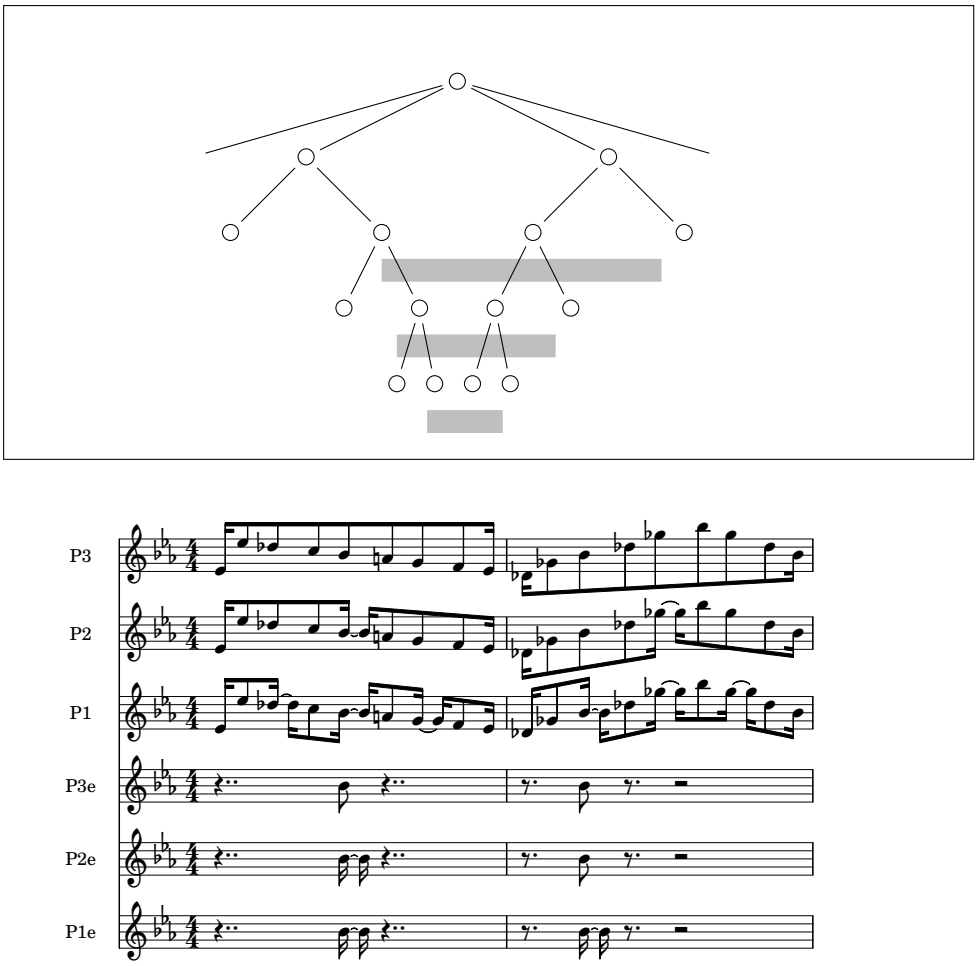


Figure 9: Anton Bruckner, Third Symphony, Adagio, Bar 137: Syncopes With Factor=2 Crossing Different Numbers of Levels of the MT, Controlled by the Style Parameter "`max_level_syncope_2 =3/=2/=1`".

child of the right sibling, etc.) and can be applied to the other direction (the right child of the left sibling, etc.)

The match for open transformations must consider conflicts and combinations. For each event, the sequence of nodes in the IC is divided into sub-sequences of maximal length which combine three segments, see Figure 10:

1. An ascending sequence of "niece and aunt", provided the niece's parent is equidistantly divided by two(2) and has the same duration as the aunt node;
2. a sequence of siblings, provided they all have the same duration;
3. a descending sequence of "nephew and uncle", as in segment 1.

The last node of one segment is shared with the following segment as its first node; each segment can denormalize to a single node. If not, a segment can possibly be represented by a note symbol as follows:

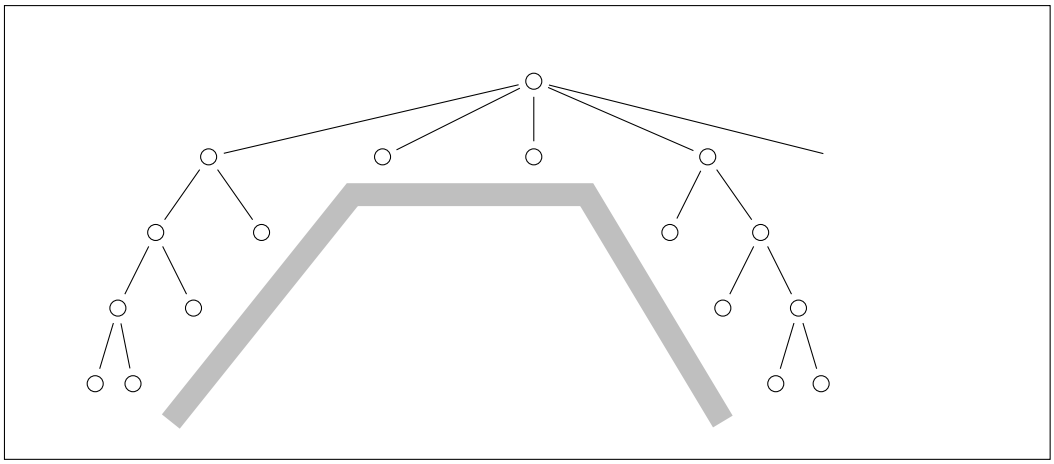


Figure 10: Pattern Matching for the Open Merging Transformations MX-D and MX-S

$$
\begin{array}{lcll}
\mathcal{M}_1 & = & 1 & \\
\mathcal{M}_2 & = & 3 & \\
\mathcal{M}_3 & = & 7 & \\
\mathcal{M}_4 & = & 15 & = \quad 3 * 5 \\
\mathcal{M}_5 & = & 31 & \\
\mathcal{M}_6 & = & 63 & = \quad 3 * 3 * 7 \\
\mathcal{M}_7 & = & 127 & \\
\mathcal{M}_8 & = & 255 & = \quad 3 * 5 * 17
\end{array}
$$

Table 2: First Eight MERSENNE Numbers and Their Prime Factors

- Segment 1 can be translated to a "negative dotting", i.e. a dotted duration symbol which stands on a "weak" metric position.
- The nodes in segment 2 can be replaced by fewer nodes of longer duration symbols.
- The nodes in segment 3 can be written as a "positive dotting", i.e. a dotted duration symbol which stands on a "strong" beat.

If two adjacent segments contain more than one node (> 1), they compete over the common node; segment 1 and 3 can be adjacent iff the middle segment denormalizes. Currently both kinds of dotting have priority over the sibling comprehension, and positive dotting has priority over negative. This could be changed by introducing further style parameters, but up to now this has not shown necessary.

2.11 E-Writability and Applicability of the MXs

All MXs are only applied if their *resulting note* (i.e. the one single note symbol representing more than one nodes from the IC) is writable as one(1) single duration symbol (e-writable). This is due to the origin of the MXs from performance practice: They want to immediately signal one typical metric/rhythmic situation. For this sake, their graphic appearance must be simple. (At least, the replacement must be much simpler than the group of notes which are replaced !-) While their concrete

appearances come from **(RqStyle)** and thus are configurable, this requirement is indispensable and comes from **(RqErgo)**. This decision violates **(PropLocal)**: Up to now the question of writability has not been considered at all; the MXs themselves have been defined based on the shape of sub-trees, not talking about concrete duration values.

It is fundamental that the writability of the *denominators* of all durations must be realized by inserting essential brackets, anyhow. Therefore here (and in the final write-out phase) only the *enumerators* must be considered. This is an important instance of the **(PropComp)** principle, which is fundamental for the design of METRICSPLIT and simplifies it significantly. So all following propositions about d- and e-writability in this section are always implicitly meant "under the current bracket stack".

At this point we assume that no additional proportions ("convenience brackets" like "4:3") have been applied. Then a duration p/q is e-writable, if and only if its enumerator has a value $p = (2^n) * (2^k - 1)$, with $n \in \mathbb{N}_0, k \in \mathbb{N}$. This follows from the definition of the prolongation dot, see above in section 2.2.

Definition 2.15 (M-Duration)**.** An *M-duration* is a rational duration p/q with p is a product of a power of 2 and a so-called MERSENNE-*number* $\mathcal{M}_k = 2^k - 1$.

$k - 1$ is the number of prolongation dots. Switching from general analysis to **(PropHeur)**, we restrict it to $k \leq 8$, i.e. maximally seven prolongation dots. The prime factors of these $\mathcal{M}_k$ can be found in Table 2; most of the $\mathcal{M}_{k \leq 8}$ are prime.

Furthermore there are different style parameters, which for different contexts put different upper limits on the number of prolongation dots, thus realizing **(RqErgo)** and **(RqStyle)**. This introduces a kind of **(PropHeur)** into to the general considerations.

Please note that the criteria of writability and the limits on prolongation dots will come into play in very different situations, following **(PropHeur)**, on a meta-level: By **Param:**`int max_dots_positive` MX-Y can be inhibited completely (see next section), but MX-D will only be modified (see section 2.13).

2.12 (Ph-MX): Algorithm

The algorithm gets a sequence of *input nodes* of length ℓ, which is the IC of a single event.

It calculates (a) the *print set*, which is a subset of the nodes in the IC and indicates which nodes will be printed (=visible), (b) a map from these nodes to their new duration values[9], and (c) a map from these nodes to adjusted beaming information. In the implementation, there are map objects which "non-invasively" override the original data of the node from the MT, which is not modified as such, but left untouched for re-use.

- If $\ell = 1$, put the single node into the print set. No further calculations are required.
- If the event is sound, not silence, then test for the closed patterns (MX-Y and MX-H) as follows:
  - If the first input node is the first subnode of its parent (technical index = 0), then the test fails.

[9] Means: The duration they represent after application of MX = their own duration plus the sum of all invisible successor nodes before the next printed one. This data is memorized for efficiency only; the information is of course totally given by the MT and the print set.

  - It the parent of the first input node is equidistantly divided by 2, and $\ell = 2$, and both input nodes have the same duration, and this duration is e-writable not using more than **Param:**`int max_dots_positive` prolongation dots, then test for a syncope-2 as follows:
    Determine the nearest common parent of both nodes in the list. If the paths from there to both nodes have the same length, and this length is $\leq$ **Param:**`int explicit_max_level_syncope_2`, and all inner nodes on these paths are equidistantly divided by two, then return that MX-Y shall be applied.
  - Otherwise, if the parent of the first node (p1) has a following sibling (p2), and both have the same duration and are equidistantly divided by the same number $n > 2$, then look for syncopes and hemiolas as follows:
  - If an EB starts with p2, and **Param:**`merge_may_cross_bracket_limit` = `false` , then the test fails. This style parameter allows syncopes/hemiolas to make n-plet-brackets "hang in the air", without a note symbol at their starting position, see line b) in Figure 4 on page 19.
  - If $n = 3$ and $\ell = 2$, and the first node in the event's list is the last subnode of p1 and the second is the first of p2, and **Param:**`hemiola` = `true` and the (identical) duration of both nodes is e-writable not using more than **Param:**`max_dots_positive` prolongation dots, then return that MX-H shall be applied.
  - Otherwise, if $n = k$ and the nodes in the list are the concatenation of a suffix of the subnodes of p1 and a prefix of the subnodes of p2, and (identical) duration of the parents is e-writable with $\leq$ **Param:**`max_dots_positive` prolongation dots, then a "long" syncope has been found. There are three different additional Boolean style parameters which control their application, namely **Param:**`syncope_2_3` for three nodes starting at measure position two(2)[10], **Param:**`syncope_3_3` for position three(3) and **Param:**`syncope_longer_4` for all longer node lists.
  - Otherwise no closed MX can be applied.

- If the test for the application of a closed MX has succeeded, then put only the first node into the print set. The sum of the durations of all nodes is memorized as its duration, and the number of the beams between p1 and p2 (there are no beamlets, because the duration is identical) is taken as the number of left and right beams of this node.
  With this, the event's processing is finished.
- If a closed pattern has been found, but the MX is not applicable due to style parameters, then matching open patterns is impossible. The event's processing is finished by putting all nodes into the print set.

The necessary criteria for the application of the closed MXs are simple and follow from the simple quotients between the durations of the nodes involved:

Proposition 2.16. *For a syncope MX-S to be applicable, the duration of the parent nodes must be an M-duration.*

Proposition 2.17. *For a hemiola MX-H to be applicable, the duration of the child nodes must be an M-duration.*

[10] Counting beats in a in measure starts with one(1)!

00000011	$3 = \mathcal{M}_2$		00000111	$7 = \mathcal{M}_3$		00001111	$15 = \mathcal{M}_4$	
×00000101	×5		×00001001	×9		×00010001	×17	
= 00001111	$= 15 = \mathcal{M}_4$		= 00111111	$= 63 = \mathcal{M}_6$		= 11111111	$= 255 = \mathcal{M}_8$	

00000011	$3 = \mathcal{M}_2$
×00010101	×21
= 00111111	$= 63 = \mathcal{M}_6$

Table 3: Quotients of Two MERSENNE Numbers

If no closed pattern has been recognized, then look for open patterns (MX-D and MX-S). For this, the prefix of the input nodes is repeatedly split into the three segments as defined above, see Figure 6:

- Start with a cursor set to the very first input node.
- (LOOP:)
  Memorize the cursor value as the start of the first segment (i_0 :=cursor).
- While the next input node is a sibling of the parent of the cursor[11], and that parent is equidistantly divided by two, and both siblings have the same duration, then advance the cursor.
- Otherwise memorize the cursor value as the start of the second segment (i_1 :=cursor).
- While the next input node is a sibling of the cursor, and both have the same duration, and (there is no EB starting at the sibling, or **Param:**`merge_may_cross_bracket_limit` = `true`), then advance the cursor.
- Otherwise memorize the cursor value as the start of the third segment (i_2 :=cursor).
- While the parent of the next input node is a sibling of the cursor, and that parent is equidistantly divided by two, and both siblings have the same duration, then advance the cursor.
- Otherwise the cursor stands at the end of the last segment (i_3 :=cursor).

These segments now have to be translated. Here is a clear "break of paradigm", and a loss of **(PropComp)**: Up to this point, only the shape of the graph and the identity of durations have been tested; from here on, concrete duration values affect the result.

The durations of the nodes in the three collected segments differ maximally by powers of two, so their d- and e-writabilities are the same. Let b be the longest of these, i.e. the duration of the nodes in the middle segment.

An MX-D is applicable, if its result is e-writable, i.e. has an M-duration. If the MX-D candidate segment (first or third segment, as constructed above) has the length k, then $\mathcal{M}_{k\geq 2}$ is a factor of the result duration's enumerator, due to the repeated quotient "1:2" between the duration of the nodes involved. The other factor comes from b. If this is simply a power of two, every k leads to an M-duration in the result.

But if there b has a largest odd factor $f > 1$, only sporadically this combines with the particular values of k to M-durations in the result, see Table 3.

((

By the way: The generative principle in the binary notation in this table shows clearly, that the quotient between two MERSENNE-numbers cannot be a MERSENNE-number again. For musical notation this means, spoken informally, that "a dotted base

[11] This implies it to be the immediately following sibling, since the nodes come from an IC.

f	dottings		$g' =$ siblings	dottings
1	$\mathcal{M}_{\_}$	*(=all dots)*	$\{1\} \cup \mathcal{M}_{\_}$	$\mathcal{M}_{\_}$
3	$\emptyset$	*(=no dots)*	$\{1, 5\}$	$\emptyset$
5	$\{3\}$	*(=one dot)*	$\{3\}$	$\{3\}$
7	$\emptyset$	*(=no dots)*	$\{1, 9\}$	$\emptyset$
9	$\{7\}$	*(=two dots)*	$\{7\}$	$\{7\}$
11	$\emptyset$	*(=no dots)*	$\emptyset^{a)}$	$\emptyset$
13	$\emptyset$	*(=no dots)*	$\emptyset^{a)}$	$\emptyset$
15	$\emptyset$	*(=no dots)*	$\{1, 17\}$	$\emptyset$
17	$\{15\}$	*(=three dots)*	$\{15\}$	$\{15\}$

[a)] do *not* merge, since result would not be e-writable.

Table 4: Possible Highest Odd Factors for MX-D and MX-S for Highest Odd Factors in the Base Duration

duration cannot be dotted."
))

For MX-S, applied to the middle segment, the conditions are a bit weaker. Let g be the number of sibling nodes to be merged, g' the highest odd factor therein, and f as above. Then it must hod that $f * g' = \mathcal{M}_{x \geq 1}$.

If one of the values b or g is a simple power of two ($f = 1$ or $g' = 1$), then for the other value ($= \square$) all powers of two times all $\mathcal{M}_x$ are possible. In the number range up to 10, which is relevant in practice, this means only

$$\square \notin \{5, 9\}$$

Otherwise, the following combinations of f and g' are possible:

3	7	15	5	3	3
5	9	17	51	85	21

These are the pairs of factors from Table 3, plus those got by shifting a prime factor from one side to the other, because the factors in the middle segment need not to be $\mathcal{M}_{\_}$ themselves.

Table 4 maps f (= the highest odd factor of the base duration, restricted to those relevant in practice) to the possible dottings and to g' (= highest odd factor of the numbers of siblings to merge).

If $f > 1$ is a $\mathcal{M}_{\_}$, then nearly all numbers of siblings can collapse to one single duration symbol, but no dotting is possible; if $f > 1$ combines with an $\mathcal{M}_{\_}$, one particular dotting and one particular odd factor for siblings is possible.

So the algorithm continues as follows:

- If the highest odd factor f in the base duration does permit neither dotting nor sibling merge, than every node in all segments are included in the print set, and no further calculation is needed.
- In this case advance the cursor and continue with matching the next three segments, by going back to (LOOP:) on page 35, as long as the last node of the IC has not yet been processed.
- If f does not permit dotting, but sibling merge, than all nodes before i_1 and after i_2 are included in the print result.

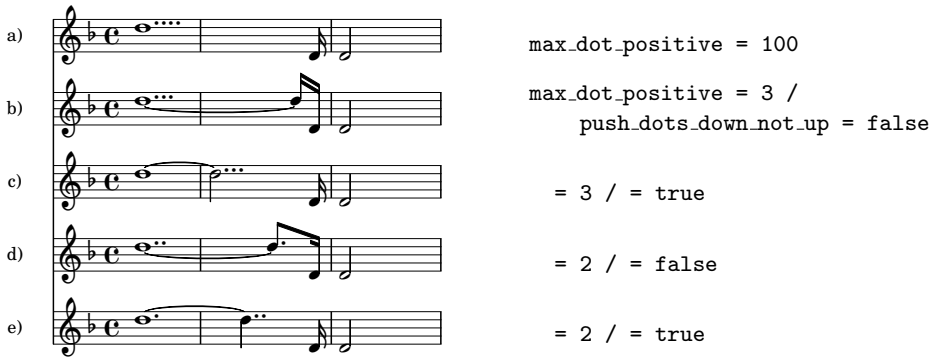


Figure 11: Different Distributions of Dots

- From the middle segment (i_1 to i_2, both inclusive), the longest prefix is taken with a length of $g' \times (2^x)$. The first node is included in the print result, and the beaming is adjusted accordingly.[12]
  The prefix is removed and the process is repeated, until the middle segment is totally swallowed.
  Then advance the cursor and continue with matching the next three segments, by going back to (LOOP:) on page 35, as long as the last node of the IC has not yet been processed.
- Last case is that dotting and sibling merge both are possible.
  If there are more than one sibling nodes ($i_1 < i_2$) or no positive dot candidates ($i_2 = i_3$), then the rightmost candidate for negative dotting is set to $i_N = i_1$, otherwise to $i_N = i_1 - 1$. This reflects the fact that positive dotting has priority.
  If there are more than one candidates for negative dotting ($i_N > i_0$), then these nodes are written out as dotted notation, as described in section 2.13.
- If there are more than one siblings ($i_1 < i_2$), then test for MX-S:
  The first candidate for sibling merge is $i_S = i_1 + 1$, if $i_0 < i_1$ (=negative dotting has been written out), otherwise $i_S = i_1$ The last candidate is $i_T = i_2 - 1$, if $i_3 > i_2$, because positive dottings have priority, otherwise $i_T = i_2$. If there are candidates left ($i_T > i_S$), then this middle segment (from i_S to i_T, including both), is transformed as described above for the previous case.
- Finally, if there are candidates for positive dotting ($i_2 < i_3$), these are written out as dotted notation, as described in section 2.13.
- Then advance the cursor and continue with matching the next three segments, by going back to (LOOP:) on page 35, as long as the last node of the IC has not yet been processed.

2.13 (Ph-MX): Distributing Prolongation Dots

The transformation MX-D is applied to the first and third segment identified in the preceding step, generation negative and positive dotted notations. The most case described here is $f = 1$, i.e. the base duration of the nodes involved is a power of two and has no odd factors. This is the most versatile case: All numbers of dots are applicable.

In contrast to the closed patterns MX-Y and MX-H, the upper limits on the dot

[12] Beaming adjustment still missing in the current implementation.

counts do not inhibit the application of this transformation totally, but modify the shape of the result. Different style parameters are respected:

- **Param:**`int max_dots_positive` how many dots are maximally allowed for a positive dotted notation.
- **Param:**`int max_dots_negative` idem, for negative dottings.
- **Param:**`int max_dots_for_pauses` is additionally respected for pauses (setting to zero disables dotting of pause symbols completely).

When the number of required dots exceeds the number permitted, more than one dotted duration symbol will be written (joined with a tie in case of sound not pause). The style parameter **Param:**`push_dots_down_not_up` = `true` indicates that the shorter duration symbols gets the maximum number of dots, and the longer symbols only the rest.[13] Figure 11 shows a famous example with different combination of style parameters.

The algorithm works as follows:

- If the style parameter **Param:**`push_dots_down_not_up` is true with negative dotting or false with positive, then set the cursor to the leftmost (=earliest) node in the segment's node list.
- Enter the node at the cursor into the print set.
- Advance the cursor by the maximally allowed number of dots plus one.
- If this is still inside the segment, and we are in a negative dotting, then copy the beaming information of the immediate predecessor of the cursor to the rightmost node in the print set (= the most recently added).
  Enter the node at the cursor into the print set.
  If the cursor position is exactly the last node of the segment, we are finished; otherwise go to the preceding step.
- Otherwise, if the cursor is outside the segment, we have to encode some *rest dot nodes*, i.e. nodes of the segment after the most recently printed node N. In case that every dot number is allowed (=there is only a maximal limit on dot numbers), the processing of the segment is complete; if we are in a negative segment, only copy the beaming information of the very last node of the segment to N.
  (The case of an additional lower limit is discussed below.)
- If the style parameter **Param:**`push_dots_down_not_up` is false with negative dotting or true with positive, then set the cursor to a virtual node behind the right end of the node list.
- Go left with the cursor by the maximally allowed number of dots plus one.
  If this position lies inside the segment, than enter this node into the print set.
  If we are in a negative dotting, then copy the beaming information of the left neighbor of the previous cursor position to this node.
  If the cursor does not stand on the very first node, then repeat this step.
- If the cursor has fallen out of the segment, then the prefix of the segment up to the last cursor position = N are again some rest dot nodes. Again, if there is no lower limit on dot counts, add the very first node = F of the segment to the print set. Additionally, if we are in a negative segment, then copy to F the beaming information of the left neighbor of N.

[13] A proposed alternative strategy to distribute the dot counts equally between two symbols had been discarded because it violates **(RqCons)**!

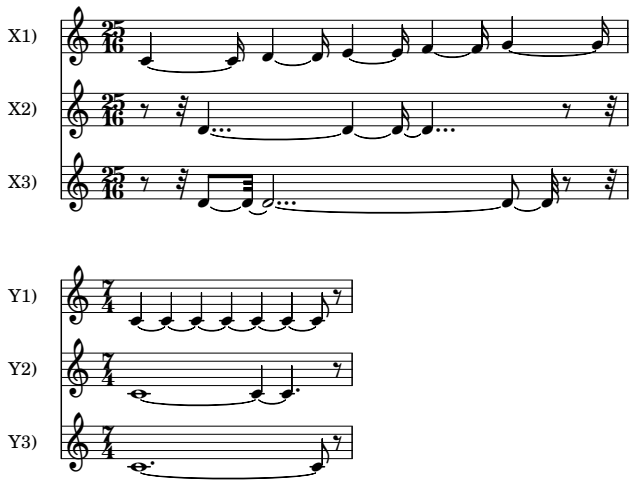


Figure 12: Switching Priority Among MX-D and MX-S

In case of negative dottings, the node(s) printed are the leftmost (=first=shortest) nodes of each tied group (for to define the measure relative start time, etc.), but the beaming is taken over from the rightmost(=latest=longest) node.

In the cases that $f > 1$ (=the maximal odd factor of the duration of all nodes), then maximally one number g' makes $f * g'$ a MERSENNE-number, as discussed above. So only one particular number of dots is applicable. The algorithm is the same as described above for $f = 1$, but the very last step: Since a dotting with fewer dots is not possible, *all* nodes of the rest dot nodes are entered into the print set, with unaltered beaming info.

(In practice this is hardly relevant, because the necessity of a double dotting would already need a base duration with odd factor 9, which is equidistantly divided by two over at least two levels of the MT, – a very special MTS!-)

2.14 Special Conflicts Mx-S Vs. Mx-D

In section 2.12, first the shape of the target pattern is stored in i_0, i_1, i_2, i_3; then from these the three segments are derived. The nodes at the "knees" i_1 and i_2 are treated according to simple and fixed priority rules. This does not yield satisfying results in some extreme cases.

(a) When $f = 1$ allows all numbers of prolongation dots, and **Param:**`push_dots_down_not_up` = `true` , and only one single node is left alone after distributing the maximum allowed dot count, then this node (at position i_1 or i_2), will be visibly written out as such, but in many cases would better be part of the MX-S in the middle segment.

(b) Even worse when $f > 1$ allows only a particular dot count > 2, and more than one un-dotted notes survive at the upper end of the first/last segment.

(c) The analog cases where these rest nodes appear at the lower end due to **Param:**`push_dots_down_not_up` = `false` do not look so ugly, but also miss the minimal possible number of note symbols in the output.

These situations are treated better when they are recognized before the segment

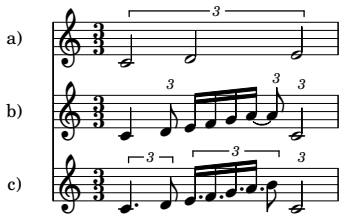


Figure 13: Adequate Level of N-Plet-Brackets Changing With Contents

limits are fixed, and switch the priority between MX-D and MX-S by setting $i_N := i_1 - 1$ and $i_T := i_2$ anyhow.

(d) The opposite case is with $f = 5$ and $(i_0, i_1, i_2, i_3) = (0, 1, 3, 4)$. Then the two MX-D (one dot) in the outer segments fit optimally, but no MX-S can happen. It could yield a better result in the sense of **(RqErgo)**, to drop the MX-Ds for the MX-S, even for the higher (=worse) number of duration symbols in the outcome, see Figure 12, lines X).

(e) Similar with $f = 1$ and $(i_0, i_1, i_2, i_3) = (0, 0, 5, 6)$. Here it would even bring one Note symbol less when giving up the dotting, see same figure, lines Y). (But this case is really rare, because it requires at least 7(seven) equidistant nodes on the same sibling level!)

But these proposed remedies give up the clean separation of the sub-phases in section 2.12, namely without and with considering f and e-writability. Therefore they are not implemented, – the cases are obviously rare (case (b) requires $f \geq 9$ and (d) a metrum like 25/16) and more of theoretical than of practical interest.

2.15 (Ph-BrSel): Selection Between Alternative N-Plet Brackets

As mentioned above in section 2.4, essential brackets (EB) for n-tuples are calculated on stock, in correspondence to the MT. The described strategy is sufficient for traditional CWN. But the rules for MTS allow much more complicated structures, e.g. "`3*(1/4+1/12)`", see Figure 13.

This is a metrum with the overall duration of 1/1, consisting of three nodes of 1/3 (Half notes under triplet bracket), each of which consists of a "normal" quarter and a triplet eighth.

Constellations like this are not possible in traditional CWN, but in some experimental contemporary styles. Here it is *not* the case that the prime factor three(3) of the `1/12` "disappears" in the parent node's duration `1/3`. So the basic assumption does not hold, which allows the calculation of EBs as described above in Section 2.4.

The adequate notation for the intention of the author of the MTS can require brackets on three different levels in the MT: A short bracket on the lowest level, spanning only the 1/12, whenever an event appears at this position, see Figure 13b). But if there is no such event, and the whole 1/3 is one single event, a bracket spanning the half note is appropriate, or even on top level.

If the longer brackets would also be used in the first case, than the 1/4 must be written by compensating the bracket with a prolongation dot, see c). (In more complex cases, even an EB could become necessary for such a compensation.)

A similar case arises when a "9:8" bracket shall be used when eighth notes are present, but a "3:2" when not.

Therefore in the next phase, a selection of the appropriate n-plet brackets seems necessary, if they can appear on different levels in the MT. This phase is not yet implemented in METRICSPLIT, and the use cases described are not yet supported.

2.16 (Ph-FS): Duration Symbol Selection and Free Sectioning

Finally the result of **(Ph-MX)** must be written out, i.e. the nodes of the MT in the print set must be translated into sequences of front-end CWN duration Symbols, namely E_p and E_s from the grammar in Definition 2.3 on page 12. Each such node represents now its own duration plus the sum of the durations of all successor nodes from the IC which are not in the print set, due to some merging transformation, up to the next printed one.

Wherever an MX has been applied, this duration is e-writable and printed as such. But with a node N which comes directly from the IC, non-e-writable durations like 5/16 can have survived. This case is treated as follows:

- If the division of N is equidistant and by two(2), then the same problem of representation would re-occur one level lower. Therefore a *Free Sectioning (FS)* is chosen as its representation.
- Otherwise, if the number of child nodes of N is larger than the style parameter **Param:**`int max_childs_to_print`, again the FS is selected.
- Otherwise, if one of the child nodes is itself not e-writable, again the FS is selected.
- Otherwise the child nodes are printed as duration symbols and these are joined by tie symbols.

A FS is a representation of the duration with more than one, but least possible duration symbols, joined by tie symbols, *not* considering an MT. (This independence is meant by "Free"). For construction, again only the enumerator e of the duration needs to be considered. Each dotted duration symbol has an enumerator of $(2^n) * (2^k - 1)$. Its *binary representation* is a closed sequence of k digits "One" ("`1`"), followed by n digits "Zero" ("`0`"). So in the binary representation of e, all maximal chains of `1` bits found therein (simply "chains" in the following) can be represented by one(1) dotted duration symbol. e can be normalized by shifting it right until the lowest bit is set. So a first candidate for the FS is found immediately, see Figure 14a.

In the minimum case of a FS there are two(2) chains.

If the higher chain and the gap between the chains both have the length one(1), then one(1) further solution can be derived: Replace the `0` bit immediately above the lower chain in both summands by a `1` bit (position K in Figure 14b) and the `1` bit in higher chain by a `0` (at position M). Then the `0` at position K and the `1` at position M will re-appear after adding both summand. (The upper chain must have length one(1) because otherwise these replacements would create a third chain at position X, and the derived variant would not be a minimal solution.)

If the gap between the chains is larger than one (> 1), the result of adding the two Ones must be carried from position K to M by inserting further `1` bits at all positions L. This can be done in either of the summands, thus three(3) different solutions exist, see c).

Due to the associativity of addition, this derivation can be applied also when three(3) chains are found initially. The minimal case is shown in d). First it can be

```
a)          b)          c)          d)
0110 0011   0000 1011   0010 0011   0001 0101

0110 0000   0000 1000   0010 0000   0001 0000
0000 0011   0000 0011   0000 0011   0000 0100
                                    0000 0001
              X MK        XML LK        MK          M LLK
            0000 0100   0001 1100  d1) 0001 0000  d3) 0000 1110
            0000 0111   0000 0111      0000 0010      0000 0100 \
                         XML LK        0000 0011      0000 0011 /
                        0000 0100        M K            M LLK
                        0001 1111  d2) 0000 1000  d4) 0000 0010 \
                                    0000 1100      0000 0100 /
                                    0000 0001      0000 1111
```

Interpreting lowest bit as 1/64 =  :

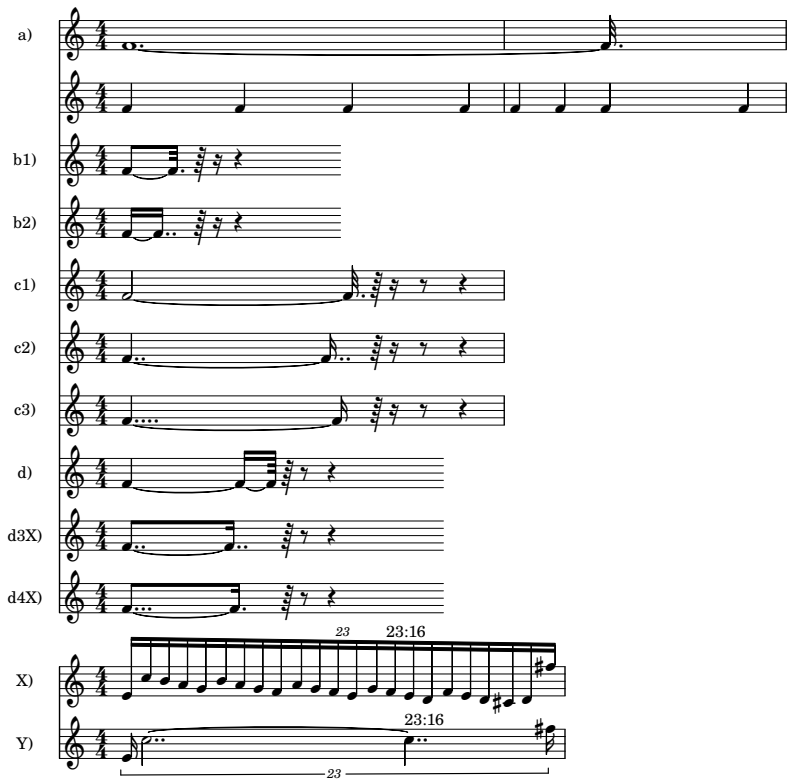


Figure 14: Construction of Free Sectionings

applied to all three(3) combinations of two(2) of the initial chains. In this minimal case, the application to neighbored chains brings only one(1) new solution, because there are no L positions, see d1) and d2). But between the lowest and the highest chain the distance is always high enough to produce two(2) new solutions, d3) and d4).

Now the same derivation can be applied again to the resulting chains, as long as the conditions are met, esp., as long as the higher one has length = 1, so to the results of d1) and d2), yielding four(4) and two(2) new variants, not shown in the Figure.

All these variants beyond the very first are *not* chosen by METRICSPLIT, for **(RqErgo)** reasons: One or more duration values appear more than once, partly hidden as prolongation dots, and thus seem harder readable. (Figure 14 shows the examples both in numeric form and in CWN notation.)

More relevance has the fact that whenever the gap between the middle and the lower chain is not larger than one (= 1), the this gap is at position K when transforming the other chains, and the middle chain and the summand which does not get the additional L-bits *collapse* to one single chain. So the initial realization by three(3) chains, as presented in d), is indeed *not a minimal solution at all*, but d3) and d4) are, when read as two(2) chains, as indicated by the bracket marks, and shown as "d3X)" and "d4X)" in note symbols.

For 21, 37, 53, 69 and 77 the initial number of chains is three(3), but by this collapsing effect their FSs need only two(2) symbols; $41 = \texttt{101001}_2$ is the lowest enumerator which indeed requires three(3) notation symbols.

Consequently, the METRICSPLIT algorithm constructs the necessary FSs according to **(PropHeur)**: It filters some practically relevant collapsing cases "hard-coded", and takes the initial, chain based, untransformed representation for all others. The higher chain = the longer duration value is put first.

Please note that this phase hardly becomes relevant, because it only treats an equidistant chain of n siblings. The most relevant case in practice is $n = 5$, but such sequences will often rely on an MT like "`3+2`" and thus *not* need FS.

A second case of examples are complex MTs like "`3*(1/8)+3*(1/8)+3*(1/8)+4*(1/8)`", where a whole measure note of 13 eighths with **Param:**`int max_childs_to_print` $\leq$ 3 will be printed as 𝅝.𝅘𝅥𝅯

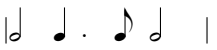
A third family of examples are CHOPINesque right-hand girlandes, which have a large number n of siblings of equal rights. Staff X1) in Figure 14 on page 42 shows such a structure (beams by LILYPOND). Y1) shows the result of some automated analysis ("highest pitch so far") which happens to need an FS of $n = 21$.

3 Comprehension, Future Work and Related Work

When constructing METRICSPLIT, it turned out that the processing phases adhere to astonishingly different processing styles. A monolithic, homogeneous algorithm seems unfeasible. Furthermore, heuristic and systematic approaches had to be mixed for practically satisfying results. To our knowledge, this is the first approach for a total function which renders arbitrary sequences of rational numbers into sensible and readable CWN.

While the current implementation supports a lot of notation variants, there are still important situations not yet covered:

- Often convenience brackets are more beneficiary w.r.t. **(RqErgo)** than their alternatives with prolongation dots and ties; their integration into the processing pipeline is a major challenge.
- Hierarchical combinations of MX are often common and sensible, e.g. applying an MX-D *after* applying an MX-H: The notation



  is currently only generated in an MT like `3*1/2`, but not in `2*3*1/4`.
- Perhaps an automated extension of the MT which adds alternatives not only *below* the user's explicit definitions, but which is also allowed to change the higher tree levels and re-combine inner nodes, is a solution to this and other cases. Also in a synthesized sequence of 17 notes, a triplet-bracket can currently only be inserted *below* the synthetic level, as shown in line d2) in Figure 4 on page 19. The solution in line d3) would be better.
- Starting with section 2.10, the discussion has been restricted to number values of practical interest, e.g. maximally seven prolongation dots. All following theorems and algorithms could be re-formulated without these restrictions. This could be of theoretic, but probably not of practical interest.
- The generation of beaming info is still under construction. This is practically relevant for generating "Music XML".
- Many small variants should be tested in practice (e.g. syncopes for pauses; implement MX-S for $f = 11$, yielding an FS; configurable penalties when constructing the stacking plan)

A discussion of the musical background and of different notation situations from practice can be found in [4] and [5] (in German language).

The earliest treatment of a similar problem is by Byrd, 1984, [2], further developed by [3]. Both approaches are partial as they only treat the transformation of a rhythm already given as sequence of note symbols into a more adequate form. Gieseking explicitly postpones the problem of "local metra", which we cover by the "alternative" edges in MT. A more important difference to ours is that they operate on a "flat" foreground model of time points and metric weights, in contrast to our MT graph. We assume this a very clear example for the difference between middle-ground and foreground phenomena.

Same problem with LILYPOND [6]: It has flexible and rather powerful configuration means for user defined beaming rules. But they also are defined on a linear scale of time points instead of a middle-ground structure, thus do not prevent inconsistencies and programming errors.

A Operation of the Demo Application

The interactive demonstration tool can be downloaded from `http://bandm.eu/metatools/download/DemoMetric.jnlp`. The deployment technology is "Java Web Start" = "javaws" = "jnlp", see [1].

It presents a GUI with several tabs. Below the tabbed field a text line shows the most recent messages from the program. The tabbed cards have these functions:

- First tab, labeled **Metric Tree**:
  In the top line the formula for the MTS is entered, according to the syntax

defined in section 2.4. Pressing return starts parsing. If successful, the resulting MT is presented in the two windows below: on the left the initial state, as defined by the MTS; in the right the MT as it evolves during the following split operations.

- Second tab, labeled **Options**:
  ... allows to specify all style parameters for all phases of MetricSplit. The radio buttons in the top line allow to select the reaction for unforeseen division factors, namely to fail or to call **(Ph-Div)** or **(Ph-App)**, see section 2.6 above. According to this selection, parts of the parameter input fields are inactive.
  (Please note that changing the style parameters will influence the next request for the metric split immediately, but the "grown" MT, shown on the right of the first tab, is *not* flushed: Its state is possibly not consistent with the new parameter settings. It can be re-initialized by re-parsing the MTS by pressing "return" in the MTS input field.)
- Third tab, labeled **Rhythm**:
  In the top input line a sequence of start points may be entered. These are simply rational numbers like "`0`" or "`2/13`", separated by whitespace.
  The second line allows to enter a sequence of duration values. Making an input in either line leads to (a) parsing the input text, (b) normalizing the data, and (c) presenting the normalized data in both input lines.
  The third line is a sequence of the characters "s" and "p", separated by whitespace, indicating whether the events are sound or pause.
  Making an (error-free) input in either of the three lines triggers the split process: The result of the splitting w.r.t. the currently selected MTS is shown in the window below. This window shows the sequence of call-backs which is performed when `MSplitter.Result.WriteOut.process()` is called, as described in section B.
  To re-eval the splitting (e.g. after style parameters have been changed), simply press "return" in one of the three input lines.
- Fourth tab, labeled **LilyPond Output**:
  When the checkbox is activated, the path to a LilyPond installation [6] must be entered in the next input line. (Full path or program name only, as entered in a command line interpreter, with no further options.)
  Then after each successful split, LilyPond source text is generated, the application is called, and the source text and the resulting score graphics are presented in the window.
- Fifth tab, labeled **Messages**:
  All messages, i.e. logs, warnings and error messages, are collected here, in one scrollable chronological list. The most recent message is shown on the single bottom line, always visible.

B Internal Structure of the Java Implementation

This section gives a short survey on the main classes of the current implementation of MetricSplit. The whole API documentation can be found at `http://www.bandm.eu/music/docs/api/eu/bandm/music/entities/package-summary.html`. A schematic survey of the most important classes and methods is in Figure 15.

For the application point of view, the topmost class is a `MetricConsumer`. It is created with a sequence of `QualifiedRational`s, which are products of rational

numbers and Boolean values, indicating "sound not silence". Either a sequence of start points or a sequence of durations can be given.

An MSplitter is created with an MTreeSpec and a collection of parameter objects. At most one of "divide" or "approximate" may be $\neq$ null , which selects **(Ph-Div)** or **(Ph-App)** for the synthesis of unforeseen division factors, see section 2.6 above.

An MTreeSpec can be constructed bottom-up by explicit constructor calls, or by parsing its external text representation with the static function parseAndInitialize(String), with the syntax described above in section 2.4.

Calling changeMetrum(MSplitter) allows to change the metrum used by the MetricConsumer with every new measure. The functions hasNext() indicates whether there are events still unconsumed, while hasCompleteNext() indicates whether these are enough to fill a whole measure.

getNext() executes the split for the next measure stores the results internally.

The typical pattern for exploiting the split results is like

```
myMetricConsumer.getNext();
myMetricConsumer.new WriteOut(){
  @Override public void open_proportion(final Rational p){
    // user code
  }
  @Override public void close_proportion(final Rational p){
    // user code
  }
  @Override public void writeOut(final int index,
                                 final DottedBaseDuration duration,
                                 final Rational prop,
                                 final StemEnd beams,
                                 final boolean isSound,
                                 final boolean isFirst,
                                 final boolean isLast){
    // user code
  }
}.process()
```

Additionally there are inquiry functions for cross measure information, like firstIsOverlap(), getLastOverlap(), isIncomplete(), etc. (The MSplitter.Result object can also be read directly, but using its WriteOut mechanism does not consider the cross measure situation correctly.)

When processing only one single measure separately, MSplitter can also be used directly. In this case the input data is also either durations or start points, but the caller is responsible for exactly covering the whole measure, see section 2.5 above.

Similar to above, the MSplitter.Result object contains a class which defines callbacks, to be overridden by the user, so the typical use is

```
final MSplitter myMSplitter
  = new MSplitter(myMTreeSpec, param0, param1, param2);
final MSplitter.Result result = myMSplitter.process(myQRats);
result.new WriteOut(){
  @Override public void open_proportion(final Rational p){
    // user code
  }
  @Override public void close_proportion(final Rational p){
    // user code
  }
  @Override public void writeOut(final int index,
                                 final DottedBaseDuration duration,
                                 final Rational prop,
                                 final StemEnd beams,
                                 final boolean isSound,
                                 final boolean isFirst,
                                 final boolean isLast){
    // user code
  }
}.process()
```

Also an `MTree` can be used directly. It is created from an `MTreeSpec` by the static function `install(MTreeSpec,MTree.Parameters)`.

References

[1] The Java Web Start Tutorial. Oracle, 1995/2015. http://docs.oracle.com/javase/tutorial/deployment/webstart/index.html.

[2] Donald Alvin Byrd. *Music Notation By Computer*, 1984.

[3] Martin Gieseking. *Code-basierte Generierung interaktiver Notengraphik*. Universität Osnabrück, 2000.

[4] Markus Lepper. Metrisch korrekte Notation von Rhythmen – Eine Einführung in das Problem. 2012. http://senzatempo.de/ston2012102700.html.

[5] Markus Lepper. Metrisch korrekte Notation von Rhythmen – Unser Lösungsvorschlag. 2013. http://senzatempo.de/ston2013120500.html.

[6] *LilyPond Music Notation*, 2011. http://lilypond.org.

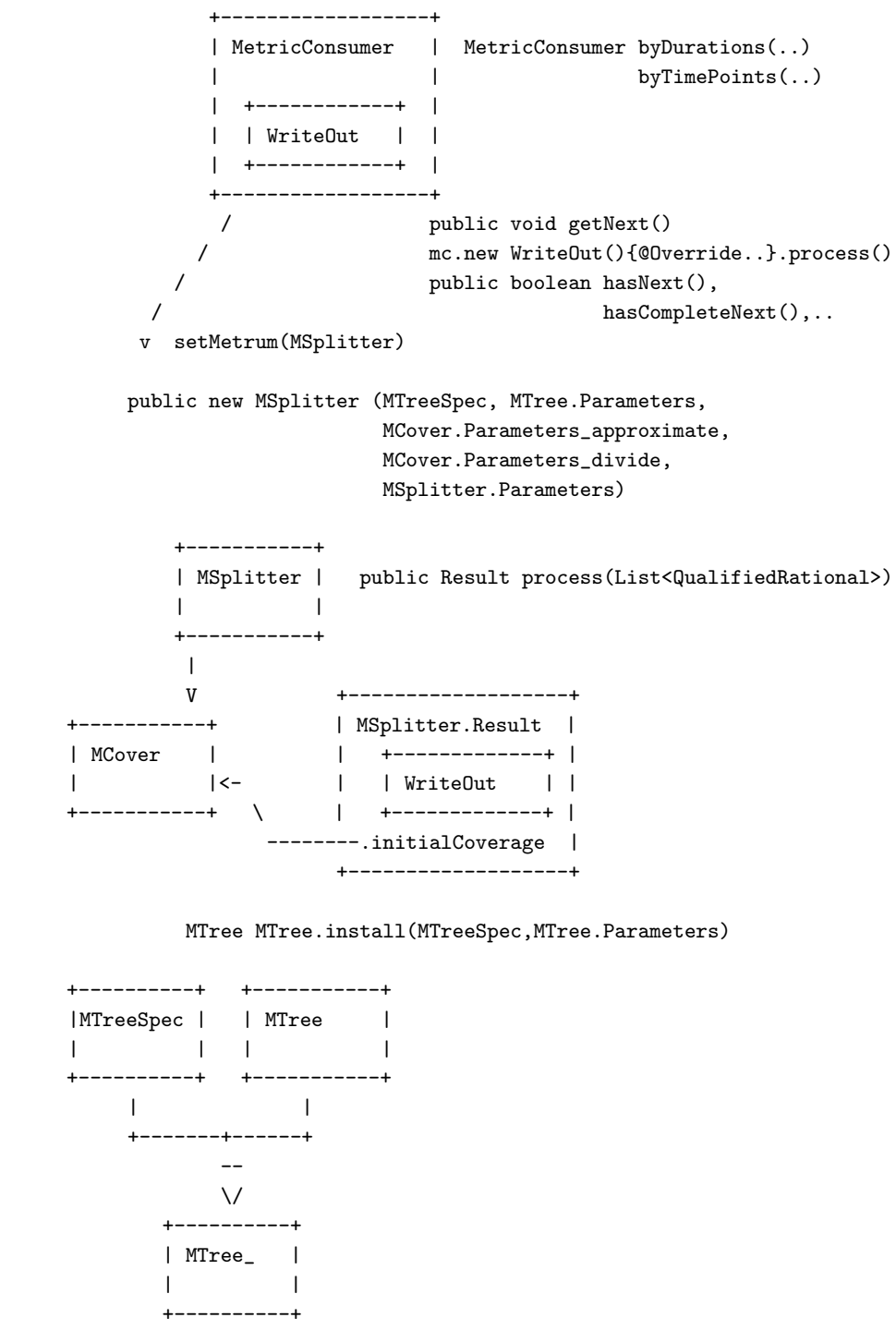


Figure 15: Classes of the Java Implementation